МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет Комп’ютерних наук, управління та адміністрування

Кафедра Інформаційних технологій

КОМПЛЕКСНА БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: Розробка мобільного застосування для керування смарт-браслетом MGCOOL BAND 4

Склад:

Частина 1 Підсистема управління інтерфейсом зв’язку і базою даних

Виконавець: Тюртюбек Євген Максимович
(П.І.Б.)

Керівник: ст.викл. Рольщиков Вадим Борисович
(П.І.Б.)

Частина 2 Розробка користувальницької частини застосування

Виконавець: Бєлодонов Олександр Сергійович
(П.І.Б.)

Керівник: ст.викл. Рольщиков Вадим Борисович
(П.І.Б.)

Староста роботи: Тюртюбек Євген Максимович
(П.І.Б.)

Провідний керівник проекту: ст.викл. Рольщиков Вадим Борисович
(П.І.Б.)

Рецензент: к.геогр.н., доцент Лужбин Анатолій Михайлович
(П.І.Б.)

ОДЕСА 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет Комп'ютерних наук, управління та адміністрування
Кафедра Інформаційних технологій

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: Розробка користувальницької частини застосування

Виконав студент 4 курсу групи К-41Б
Спеціальність 122 Комп'ютерні науки
Бєлодонов Олександр Сергійович

Керівник ст.викл.
Рольщиков Вадим Борисович

Консультант к.геогр.н., доцент
Кузніченко Світлана Дмитрівна

Рецензент к.геогр.н., доцент
Лужбін Анатолій Михайлович

Одеса 2020

ЗМІСТ

Скорочення та умовні позначки 6
Вступ.. 7
1 Аналіз предметної області.. 9
Опис мобільної платформи Android 9
Аналіз документації смарт браслету 10
2 Обґрунтування вибору технічних засобів 13
Вибір інтегрованого середовища розробки 13
Вибір шаблону проектування.. 14
Вибір системи контролю версій.. 15
2.5 Вибір засобу для створення діаграм... 19
3 Розробка графічної складової застосування.. 21
Розробка форми вибору пристрою для підключення 21
Розробка форми відображення основних даних приладу. 25
Розробка форми відображення збережених даних 30
Розробка форми керування параметрами застосування............................... 36
Розробка форми планування годинників 40
4 Розробка допоміжних модулей... 45
Розробка програмного модуля прослуховування станів.............................. 45
Розробка програмного модуля для передачі повідомлень........................... 49
Розробка маніфестного модуля.. 53
Висновки 56
Перелік джерел посилання.. 57
Додаток А Представлення основних графічних форм.................................... 59
Додаток Б Лістинг програмного коду алгоритму синхронізації..................... 65

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧКИ

ADT (Android Development Tools) – інструменти розробки Android.

ANDK (Android Native Development Kit) – набір засобів для розробки програмного забезпечення для платформи Android.

API (Application Programming Interface) – інтерфейс прикладного програмування.

BLE (Bluetooth Low Energy) – Bluetooth з пониженим енергоспоживанням.

IDE (Integrated Development Environment) – інтегроване середовище розробки.

MVP (Model View Presenter) – Модель Вигляд Представлення, шаблон проектування.

SDK (Software Development Kit) – набір засобів для розробки програмного забезпечення.

UML (Unified Modeling Language) – уніфікована мова моделювання.

ВСТУП

Програмне забезпечення для смарт браслетів на мобільні пристрої є невід'ємною частиною експлуатації смарт браслетів. Уся система складається з самого смарт браслету, котрий сам по собі має своє апаратне забезпечення та з програмного забезпечення для синхронізації даних.

У даній частині проекту буде викладені та описані процеси розробки програмної частини цієї системи, а саме застосування на Android. Будуть реалізовані абстрактні алгоритми, які надаються розробником приладу (у даному випадку набором команд та алгоритмів, які виконані у іншій частини цього проекту).

Для виконання цих завдань необхідно детально дослідити документацію для розробника, мати навички проектування та розробки Android додатків, мати навички роботи з необхідними інструментами, узгодити спільну мову програмування або найти альтернативу та мати навички розробки програмних продуктів у команді.

Програма для синхронізації має використовувати канал зв'язку, який спроектував розробник та який підтримується приладом для якого розробляється застосування, також реалізовуватися усі функції, які підтримує смарт браслет. У випадку, якщо функції мобільного пристрою можуть розширювати функціонал, то їх необхідно реалізувати.

Для того, щоб задовільнити потребу у командній розробці необхідно розуміти та знати примітивні знання що патернів розробки, наприклад MVP. У даному проекті частина Model буде повністю виконана у іншій частині проекту, незначна частина Presenter та частина View буде виконана повністю у даній частині проекту.

Розробник мобільного застосування не зобов'язаний мати представлення щодо структури апаратного забезпечення пристрою смарт браслету та тим паче щодо його фізичної структури. Таким чином достатньо виконувати вказівки куратору проекту та правильно інтерпретувати розроблену доку-

ментацію. Сам процес роботи завдяки патернам командної може відбуватися відносно незалежно, якщо увесь проект було детально сплановано. Також завдяки хмарним сервісам та технологіям процес розробки можна виконувати інспекції коду, відстежувати статистику розробки та резервувати розроблений код.

пеки операційної системи. Наприклад необхідність отримувати у користувача додатковий дозвіл для доступу до середовища даних або до стану телефону.

У цілому мобільний пристрій виконує функції зберігання та обробки отриманих даних, передачі команд та синхронізація часу з смарт браслетом. Також потужним інструментам є повідомлення, які можуть надходити з різних додатків у ідеалі їх необхідно відправляти на смарт пристрій. Також мобільний пристрій може приймати виклики та їх аналогічно можна транслювати до смарт пристрою. Таким чином застосовуючи функції мобільної платформи можна розширити функціонал смарт браслету.

Для реалізації функції зберігання та обробки даних необхідно використовувати локальну базу даних. Технологія SQLite задовольняє цим потребам. Дана технологія відмінно підходить до зберігання однотипних даних, підтримує різні платформи та може виконуватись локально.

1.2 Аналіз документації смарт браслету

Згідно отриманих вказівок та документації з першої частини даного проекту необхідно реалізувати керування пристроєм для керування існують такі команди усі ці команди описані у класі ComandIntepreter:

- команда EraseDataHeader очищення пам’яті пристрою;
- команда RestoreCommandHeader скидання до початкових налаштувань;
- команда TimeSyncHeader для синхронізації часу;
- команда GetMainInfo для вилучення даних;
- команда HRRealTimeHeader для вилучення даних у режимі реального часу;
- команда ShortMessageHeader для виведення короткого повідомлення;
- команди LongMessageHeaderPartOne та LongMessageHeaderPartTwo для виводу довгого повідомлення;

– команди StopLongAlarmHeader для припинення виводу довгого повідомлення;
– команда AlarmHeader для налаштування будильнику;
– команда GyroActionCommandHeader для налаштування реагування на зміну вертикального положення;

Важливо уточнити, що клас ComandInterpreter містить не тільки команди для керування пристроєм, а також алгоритми ідентифікації повідомлень, які надходять від пристрою. Таким чином не має потреби вивчати пристрій, а лише використовувати надані класи та рекомендації.

Також були отримані таки типи даних (класи), які описують примітивні логічні одиниці предметної області або статичні класи, які просто мають функції, які необхідні для спрощення розробки або перетворень даних:

– модифіковане сповіщення (CustomNotification);
– таблиця годинників (AlarmsTable);
– годинник (AlarmProvider);
– допоміжні утиліти бази даних (CustomDatabaseUtils);
– реакція на команди (CommandCallbacks);
– таблиця записів даних серцебиття (HRRecordTable);
– запис серцебиття (HRRecord);
– таблиця загальних записів (MainRecordsTable);
– загальний запис (MainRecord);
– таблиця фільтрів (NotifyTableFilter);
– таблиця записів циклу сну (SleepRecordsTable та SleepSessionsTable).

У цьому списку усі дані інтуїтивно зрозумілі та очевидні крім класів CustomDatabaseUtils та CommandCallbacks. Якщо усі інші це просто групування примітивних типів int або string та інших з методами вилучення або збереження, то ці два класи є статичним, які просто об’єднують набір функцій.

Клас з назвою CustomDatabaseUtils має команди для перетворення даних між отриманими у чистому вигляді від пристрою або перетворені

усередині застосування до типу даних, який може інтерпретувати база даних та навпаки.

Клас під пунктом з назвою CommandCallbacks більш схожий на інтерфейс, який необхідно реалізувати, щоб оброблювати дані, які надходять з смарт браслету.

Таким чином без зайвих проблем можна дослідити розроблений код у першій частині проекту та використовуючи документацію виконати інтеграцію даних класів до Android застосування. Даний код виконаний на мові програмування Java може бути перетворений до мови програмування Kotlin.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

У даному розділі будуть детально розглянути проблеми та завдання, які необхідні виконати у даній частині проекту.

Необхідно використовуючи документацію та вказівки виконати реалізацію алгоритмів для синхронізації даних з пристрою та його керування. Використовуючи здатності операційної системи на якому виконується застосування необхідно розширити функціонал застосування.

Опис мобільної платформи Android

Згідно з планом цільовою платформою для розробки було обрано операційну систему Android. Також відомо, що пристрій використовує технологію BLE, таким чином для підтримки застосування необхідно мати версію Android 4.3 (версія API 18) [1]1).

Засновуючись на даній інформації можна забути про підтримку додатком версій API меншої ніж 18. Згідно з даними статистики [2]2) станом на 7 травня 2019 року це втрата підтримки 3,3% пристроїв. Також важливо зазначити, що пристрій може не мати BLE фізично, навіть якщо має версію API 18 чи більше. Очевидно, що втрата 3,3% не є суттєвою проблемою, а втрата по причині відсутності підтримки зі сторони розробників мобільних пристроїв повністю знімає відповідальність та необхідність перегляду введення підтримки для даних пристроїв.

Більшою проблемою для підтримки є підтримка більш нових версій, а саме версій 5.0 (версія API 21) та вище. Причиною цьому є підвищення без-

---

1)[1] Bluetooth low energy overview | Android Developers URL: https://developer.android.com/ guide /topics/ connectivity/ bluetooth-le. (дата звернення 03.03.2020)

2) [2] Історія версій Android – Вікіпедія. URL: https://ru.wikipedia.org/wiki/ %D0%98%D1%81%D1%82%D0%BE%D1%80%D0%B8%D1%8F_ %D0%B2%D0%B5%D1%80%D1%81%D0%B8%D0%B9_Android. (дата звернення 03.03.2020)

2 ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНІЧНИХ ЗАСОБІВ

Вибір інтегрованого середовища розробки

При проектуванні даного проекту було обрано інтегроване середовище розробки Android Studio. Дане рішення було прийнято при урахуванні наявних ресурсів та навичок розробників проекту та відсутності можливостей для компіляції та запуску застосування під операційні системи IOS.

Android Studio – це інтегроване середовище розробки (IDE) для роботи з платформою Android, анонсована 16 травня 2013 року на конференції Google I / O.

IDE перебувала у вільному доступі починаючи з версії 0.1, опублікованій в травні 2013, а потім перейшла в стадію бета-тестування, починаючи з версії 0.8, яка була випущена в червні 2014 року. Перша стабільна версія була випущена в грудні 2014 року, тоді ж припинилася підтримка плагіну Android Development Tools (ADT) для Eclipse.

Android Studio [3][1)] заснована на програмному забезпеченні IntelliJ IDEA від компанії JetBrains, офіційний засіб розробки Android додатків. Дане середовище розробки доступний для Windows, OS X і Linux. 17 травня 2017 на щорічній конференції Google I / O, Google анонсував підтримку мови Kotlin, використовуваного в Android Studio, як офіційної мови програмування для платформи Android на застосування до Java і C ++.

Нові функції з'являються з кожною новою версією Android Studio. На даний момент доступні наступні функції:

- розширений редактор макетів: WYSIWYG, здатність працювати з UI компонентами за допомогою Drag-and-Drop, функція попереднього перегляду макета на декількох конфігураціях екрану;
- збірка додатків, заснована на Gradle;
- різні види збірок і генерація кількох .apk файлів;
- рефакторинг коду;

[1)] [3] Android Studio – Вікіпедія. URL: https://ru.wikipedia.org/wiki/Android_Studio. (дата звернення 05.03.2020)

– статичний аналізатор коду (Lint), що дозволяє знаходити проблеми продуктивності, несумісності версій і інше;
– вбудований ProGuard і утиліта для підписування додатків;
– шаблони основних макетів і компонентів Android;
– підтримка розробки додатків для Android Wear і Android TV;
– Android Studio 2.1 підтримує Android N Preview SDK, а це значить, що розробники зможуть почати роботу зі створення програми для нової програмної платформи;
– починаючи з Platform-tools 23.1.0 для Linux виключно 64-розрядна;
– в Android Studio 3.0 будуть по стандарту включені інструменти мови Kotlin засновані на JetBrains IDE.

Вибір шаблону проектування

Згідно з попереднім плануванням проекту було обрано шаблон проектування MVP (Model-View-Presenter) [4][1)] з причини відносної простоти користування та його сутності у очевидному розділені меж відповідальності між розробниками.

Model-View-Presenter (MVP) – це шаблон проектування, похідний від MVC, який використовується в основному для побудови призначеного для користувача інтерфейсу.

Елемент Presenter в даному шаблоні бере на себе функціональність посередника (аналогічно контролеру в MVC) і відповідає за управління подіями призначеного для користувача інтерфейсу (наприклад, використання миші) так само, як в інших шаблонах зазвичай відповідає уявлення.

MVP – це шаблон проектування призначеного для користувача інтерфейсу, який був розроблений для полегшення автоматичного модульного тестування і поліпшення розподілу відповідальності в презентаційної логіці (відділення логіки від відображення):

[1)] [4] MVP – Вікіпедія. URL: https://ru.wikipedia.org/wiki/Model-View-Presenter. (дата звернення 06.03.2020)

Модель (англ. Model) – дані для відображення;

Вид (англ. View) – реалізує відображення даних (з Моделі), звертається до Presenter за оновленнями, перенаправляє події від користувача в Presenter;

Представник (англ. Presenter) – реалізує взаємодію між Моделлю і Видом і містить в собі всю бізнес-логіку; при необхідності отримує дані зі сховища і перетворює для відображення у View.

Зазвичай екземпляр Виду (Подання) створює екземпляр Представника, передаючи йому посилання на себе. При цьому Представник працює з Видом в абстрактному вигляді, через його інтерфейс. Коли викликається подія Уявлення, воно викликає конкретний метод Представника, що не має ні параметрів, ні значення, що повертається. Представник отримує необхідні для роботи методу дані про стан призначеного для користувача інтерфейсу через інтерфейс Виду і через нього ж передає в Вид дані з Моделі.

Вибір системи контролю версій

Згідно з попереднім плануванням було обрано систему керування Github. Дана система відкрита для кожного розробника у мережі та інтегровано у багатьох найкрупніших IDE.

GitHub [5][1)] – це найбільший веб-сервіс для хостингу IT-проектів і їх спільної розробки.

Веб-сервіс заснований на системі контролю версій Git і розроблений на Ruby on Rails і Erlang компанією GitHub, Inc (раніше Logical Awesome). Сервіс безкоштовний для проектів з відкритим вихідним кодом і з 2019 року для невеликих приватних проектів, надаючи їм всі можливості (включаючи SSL), а для великих корпоративних проектів пропонуються різні платні тарифні плани.

Розробники сайту називають GitHub «соціальною мережею для розробників».

[1)] [5] Github – Вікіпедія URL: https://ru.wikipedia.org/wiki/GitHub. (дата звернення 07.03.2020)

Крім розміщення коду, учасники можуть спілкуватися, коментувати правки один одного, а також стежити за новинами знайомих.

За допомогою широких можливостей Git програмісти можуть об'єднувати свої репозитарії – GitHub пропонує зручний інтерфейс для цього і може відображати внесок кожного учасника у вигляді дерева.

Для проектів є особисті сторінки, невеликі Вікі і система стеження за вадами та багами. Прямо на сайті можна переглянути файли проектів з підсвічуванням синтаксису для більшості мов програмування. Можна створювати приватні репозитарії, які будуть видні тільки вам і вибраним вами людям. Раніше можливість створювати приватні репозитарії була платною. Є можливість прямого додавання нових файлів в свій репозитарій через веб-інтерфейс сервісу.

Код проектів можна не тільки скопіювати через Git, а й завантажити у вигляді звичайних архівів з сайту. Крім Git, сервіс підтримує отримання і редагування коду через SVN і Mercurial.

Вибір шаблонів програмування

Окрім глобального шаблону проектування, котрий стосується усього проекту необхідно обрати локальні шаблони програмування [6][1], котрі будуть використовуватись для проектування кожного класу. Професіональне та вміле використання спрощує читання коду іншими розробниками та підвищує ефективність злиття або інтеграцію з іншими частинами коду. Існують такі найвідоміші шаблони програмування: абстрактна фабрика, будівник, фабричний метод, одиночка, прототип, об’єктний пул, мультитон. Наприклад при отримані алгоритмів з іншої частини проекту зазвичай використовується шаблони програмування одиночка або пул об’єктів.

У свою чергу шаблони підрозділяються на породжувальні, структурні та поведінкові.

---

[1] [6] Джейсон Мак-Колм Смит. Елементарні шаблони проектурования = Elemental Design Patterns. – М.: «Вильямс», 2012. – 304 с. – ISBN 978-5-8459-1818-5.

Абстрактна фабрика – це породжувальний шаблон програмування сутність якого складається з класу, який представляє собою інтерфейс для створювання компонентів системи. На практиці можна використовувати для створення великої величини однотипних об'єктів.

Будівник – це породжувальний шаблон програмування сутність якого складається з класу, який представляє собою інтерфейс для створення складного об'єкту. На практиці роботу даного шаблону можна побачити при програмуванні у бібліотеках Java. Наприклад, у Android Studio для створення повідомлення користувачу необхідно використовувати «будівник» повідомлення.

Об'єктний пул – це породжувальний шаблон програмування, сутність якого складається з класу, який представляє собою інтерфейс для роботи з набором ініціалізованих та готових до роботи об'єктів. У даному проекті такими класами є усі класи, які реалізують доступ до бази даних.

Прототип – це породжувальний шаблон програмування, сутність якого складається з класу, який визначає прототип створювання об'єкту через клонування іншого об'єкту замість створювання через конструктор. На практиці використовування даного шаблону можна побачити у іграх, де повторюються однакові об'єкти.

Мультитон – це породжувальний шаблон програмування, сутність якого складається з класу, який створює іменовані екземпляри та виступає глобальною точкою для доступу для них.

Одиночка – це породжувальний шаблон програмування, сутність якого складається з класу, який гарантовано має тільки один екземпляр. У даному проекті даний шаблон використовується для класів, які реалізують певні алгоритми або сервіси, дублювання яких не передбачено.

З структурних шаблонів можна виділити адаптер, міст, компоновщик, декоратор, фасад, єдину точку входу, пристосованця, заступника.

Адаптер – це структурний шаблон програмування, який забезпечує роботу між різними несумісними класами. У даному проекті використовую-

ться для перетворення різних типів даних. Наприклад строковий тип даних з браслету у нормальний тип даних дати/часу.

Компоновщик – це структурний шаблон програмування, сутність якого є об’єднання однотипних об’єктів.

Пристосуванець – це структурний шаблон програмування, сутність якого складається у використання одного класу у різних підсистемах програми.

Заступник – це структурний шаблон програмування, який є посередником між двома другими об’єктами, який реалізує доступ до об’єкту. У даному проекті таким класом по сутності є клас, який виконує оброку команд для пристрою.

Також існують поведінкові шаблони [7][1)] з яких можна виділити інтерпретатор, посередник, ітератор, нульовий об’єкт, специфікацію, стратегію, команду, стан та відвідувача.

Інтерпретатор – це поведінковий шаблон, який вирішує завдання, яке часто повторюється, але може незначно змінюватися. У даному проекті таким класом є інтерпретатор команд, який отримує та відправляє команди до смартфону або смарт браслету.

Нульовий об’єкт – це поведінковий шаблон, який надає об’єкт «за замовчуванням» при відсутності необхідних даних.

Посередник – це поведінковий шаблон, який описує взаємодію множини об’єктів формуючи при цьому слабку зв’язаність та робить об’єкти незалежними друг від друга.

Команда – це поведінковий шаблон, який надає дію. Наприклад найменування дії та його параметри. У даному проекті такий шаблон можна побачити у інтерфейсі інтерпретатору команд. Інтерфейс необхідно реалізувати, а саме прописати реакцію на кожні дії (повідомлення від браслету до смартфону).

Стратегія – це поведінковий шаблон програмування, який необхідний при великої кількості розгалужень з великою глибиною.

[1)] [7] Мартин Фаулер. Шаблони корпоративних застосувань. М.: «Вильямс», 2012. – 544 с. – ISBN 978-5-8459-1611-2

Відвідувач – це поведінковий шаблон програмування, сутність якого у описі ситуацій, які виконуються над об'єктами інших класів.

Таким чином у даному проекті було отримано класи, які виконані за допомогою шаблонів пул об'єктів, які надають доступ до даних з бази даних. Класи-одиночки, які реалізують алгоритми зв'язку або допоміжні. Клас-заступник, клас-інтерпретатор та інтерфейс команд, який реалізує зв'язок з пристроєм.

У даній частині проекту необхідно використати шаблони для агрегації вже розроблених модулей, а саме посередник, заступник та відвідувач.

Заступниками або відвідувачами будуть самі активиті (форми) застосування, які реалізують доступ до даних. Посередниками будуть внутрішні класи для підвищення ефективності агрегації програмних модулей.

Вибір засобу для створення діаграм

Для успішного складання документації необхідно використовувати засоби для створення діаграм для моделювання. Для складання документації та діаграм у рамках даного проекту було прийнято рішення використовувати новішу версію UML 2.5.1 [8][1)], була прийнята у грудні 2017 року.

UML – це уніфікована мова моделювання для графічного опису об'єктного моделювання у області розробці програмного забезпечення, для моделювання бізнес-процесів, системного проектування та відображення організаційних структур.

UML є мовою широкого профілю та є відкритим стандартом, який використовує графічні визначення для створення абстрактної моделі системи, які називаються UML-моделлю. Дана мова проектування була створена для визначення, візуалізації і документування зазвичай програмних систем. UML не є мовою програмування, але на основі UML-моделей можлива генерація програмного коду.

[1)] [8] Про уніфіковану мову моделювання версія 2.5.1. URL: https://www.omg.org. (дата звернення 10.03.2020).

Засіб, який відмінно підходить для виконання даних потреб є онлайн-сервіс Draw.io. Даний сервіс має можливість створювання діаграм з різними стандартами та експортувати на локальний пристрій.

Також при необхідності можна використовувати вбудований модуль у інтегрованому середовищі для реверс інжинірингу розробленого програмного коду.

3 РОЗРОБКА ГРАФІЧНОЇ СКЛАДОВОЇ ЗАСТОСУВАННЯ

У мобільному застосуванні оперує два визначення: форма (activity) та клас. Форма – це опис графічної складової для відображення до користувача. Клас – це програмний код, який описує логіку та інструкції до операційної системи Android. Для кожної форми існує відповідний клас та xml розмітка з елементами графічного дизайну.

Розробка форми вибору пристрою для підключення

Згідно з алгоритмом для початку необхідно встановити з'єднання з смарт браслетом та зберігати дані цього підключення у майбутньому. Підключення до пристрою в ідеалі відбувається лише один раз, тому з даною формою користувач буде взаємодіяти лише після встановлення застосування. Для того щоб взаємодіяти з пристроєм необхідно встановити з'єднання на фізичному рівні згідно моделі OSI, а після передавати команди (прикладний рівень).

Для виконання даного завдання необхідно створити графічну форму з переліком знайдених пристроїв з можливістю обрати необхідний пристрій. У даному проекті графічна форма описана у файлі activity_scan.xml, а програмна логіка описана у файлі ScanActivity.kt у відповідному класі. Даний клас пов'язаний з класом ScannerCallback, який існує для розширення алгоритмічних можливостей класу ScanActivity та для логічного розділення структури проекту. UML-діаграми, які пов'язані з першою формою наведені на рис. 1.

Клас DeviceAdapter є класом, який перетворює дані щодо знайдених пристроїв у графічну форму, та повертає результат у клас ScanActivity при запиті за допомогою функції getView.

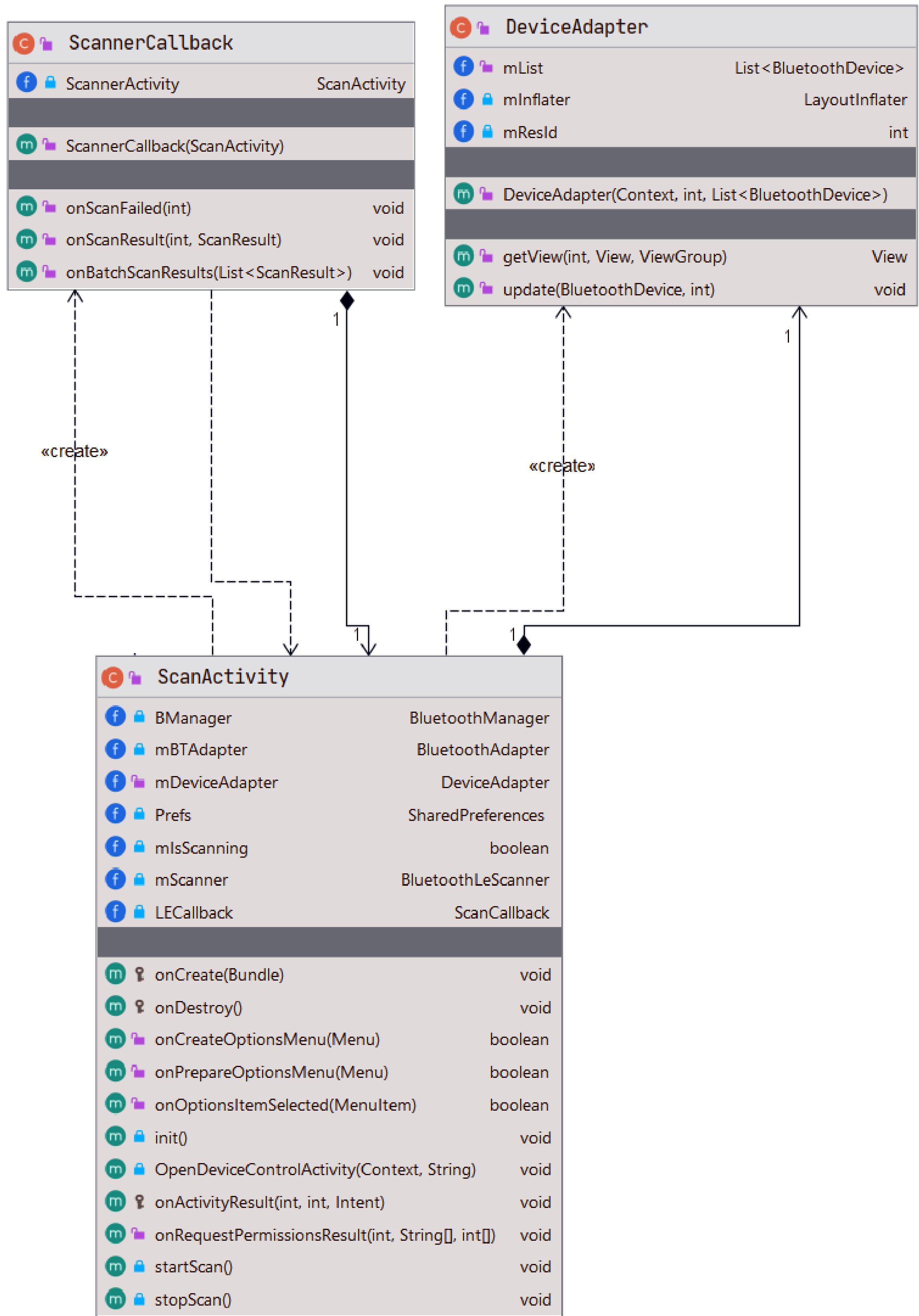

Рисунок 1 – UML-діаграма для першої форми

Клас ScanActivity має вбудовані властивості для керування апаратним забезпеченням Bluetooth смартфону через засоби операційної системи Android. Ці завдання виконують такі поля:

– BManager – системний клас, для керування технологією Bluetooth через операційну систему Android;
– mBTAdapter – системний клас для безпосереднього керування Bluetooth;
– mDeviceAdapter – розширений клас для взаємодії з просканованими пристроями;
– mScanner – системний клас для пошуку пристроїв з технологією BLE;
– LECallback – клас, який реалізує та перевантажує вбудований клас ScanCallback. Описує послідовність дій, яку необхідно виконати після сканування.

Дані поля ініціалізуються при виклику функції onCreate. Дана функція викликається автоматично операційною системою, перед тим як відобразити графічне представлення до користувача. Звичайно, якщо телефон немає необхідних технологіï, а саме BLE, то одно з ключових полей не буде проініціалізовано, а користувач отримає повідомлення, що його смартфон не задовільняє потреби проекту.

Поле Prefs є об’єктом системного класу, який реалізує можливості запису налаштувань застосування. Використовується для збереження даних пристрою, щодо якого відбувається підключення. Якщо це поле має значення, то підключення відбувається автоматично та відбувається перехід до другої форми.

Функції startScan та stopScan запускають чи зупиняють процес пошуку BLE пристроїв у полі зору.

Функції onCreateOptionsMenu та onPrepareOptionsMenu викликаються автоматично операційною системою при створені графічного вікна. Відповідають за відображення меню у верхньому правому кутку застосування.

Функція onOptionsItemsSelected відповідає за обробку подій, які пов'язані з меню у верхньому правому кутку графічного вікна.

Функції onActivityResult та onRequestPermissionsResult відповідають за обробку запитів дозволу використання ресурсів Android, які захищаються розробниками операційної системи. Згідно з правил безпеки, які встановлені розробниками операційної системи необхідно запитувати дозвіл при вмиканні адаптеру Bluetooth, сервісів місце знаходження та доступу до зовнішнього файлового сховища.

Клас ScannerCallback є необхідною складовою для запуску сканування для методу startScan. Функція onScanFailed описує алгоритм при невдалій спробі встановити послідовність сканування. Спадкує системний клас ScanCallback.

Функція onScanResult запускається при знаходженні BLE-сканером сумісного пристрою. Підтримує асинхронний режим роботи.

Функція onBatchScanResults запускається при завершені послідовності пошуку BLE-сканером через заданий проміжок часу або по запиту. Не підтримує асинхронний режим роботи та зазвичай використовується в послідовних алгоритмах.

Клас DeviceAdapter є розширенням системного класу ArrayAdapter, який створений для того, щоб відображати певний елемент у списку елементів. У даному випадку список елементів – це проскановані та знайдені прилади, а кожна одиниця в цьому в списку – це знайдений прилад з унікальною MAC-адресою. Спадкує системний клас ArrayAdapter.

Функція getView є системною. Описує спосіб відображення елементів у списку. Викликається операційною системою.

Функція update виконує додавання чи оновлення елементів у списку. Викликається у функції onScanResult у класі ScanCallback. Розроблена під асинхронний алгоритм.

Розробка форми відображення основних даних приладу.

Після підключення до приладу та ідентифікації, що це дійсно є смарт-браслет MGCOOL BAND 4 або інший сумісний, то необхідно встановити зв’язок та відобразити дані, також необхідно запустити «демона» (комп’ютерна програма у системах класу UNIX, яка запускається системою та працює у фоновому режимі без прямої взаємодії з користувачем) також відомі як фонові сервіси. Після виконання цих етапів застосування починає свою повноцінну роботу.

UML-діаграма відображує лише зв’язки з класом DeviceControllerActivity, дочірніми елементами зв’язків першого порядку можна знехтувати у даному контексті.

Клас ComandInterpreter та інтерфейс CommandReaction виконані та задокументовані у першій частині проекту. У даній частині проекту дані класи використовуються для виконання взаємодії з пристроєм.

Зовнішній вигляд другої форми описаний у файлі activity_device_controller.xml, програмну логіку описано у файлі DeviceControllerActivity.kt. Даний клас має велику кількість взаємодій та є ключовим у даному застосуванні. На рис. 2 відображена UML-діаграма класу DeviceControllerActivity з описом ключових взаємодій.

Важливо підкреслити, що дана форма взаємодіє з сутностями CommandInterpreter та CommandCallback, які були розроблені у першій частині, та дана форма відповідає за породження класу Algorithm, який виконує основне управління смарт-браслетом.

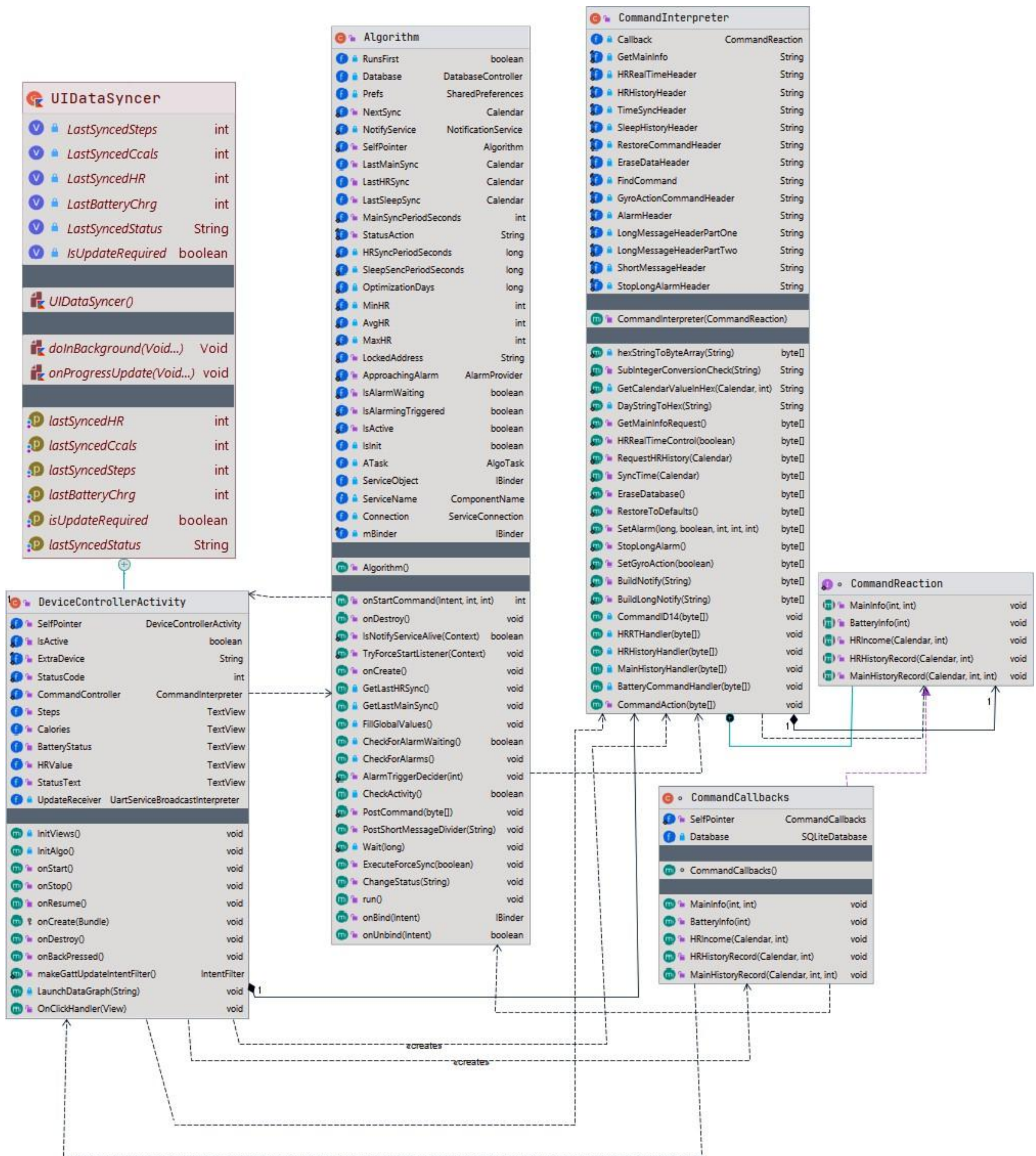

Рисунок 2 – UML-діаграма другої форми з описом ключових взаємодій

Клас DeviceControllerActivity є програмною логікою другої форми. Поля Steps, Calories, BatteryStatus, HRValue, StatusText є об’єктами класу TextView (тексту) та відображують на дисплеї користувача синхронізовані значення кількості шагів, витрачених калорій, статусу акумулятора, значення

серцебиття за останні п’ять хвилин та статус застосування відповідно. Має такі поля та методи:

- поле CommandController є фізичним об’єктом класу CommandInterpreter, а точніше реалізацією вкладеного до нього інтерфейсу CommandReaction. Реалізація даного інтерфейсу є відгук на події отримання інформації з пристрою;
- поле IsActive та StatusCode відповідають за визначення статусу даного вікна. Перше показує чи працює застосування у цілому, друге – закодований у цілочисельне значення статус. Наприклад чи є зв’язок з смарт-браслетом;
- поле instance вказує на об’єкт даного класу для виконання прямих змін до інтерфейсу. Це поле є статичним;
- метод InitViews відповідає за ініціалізацію графічних елементів;
- метод InitAlgo відповідає за ініціалізацію демонів;
- метод ReInit відповідає за алгоритм перезапуску при наявності вже запущених демонів;
- метод LaunchDataGraph відповідає за запуск третьої форми, яка відображує зібрані дані у табличному виді;
- метод OnClickHandler описує алгоритми для усіх областей на графічному вікні на які є можливість натиснути;
- метод makeGattUpdateIntentFilter() повертає зібраний фільтр, який використовується при передачі внутрішніх повідомлень усередині застосування використовуючи сервіси передачі повідомлень Android;
- методи onStart, onResume, onStop, onCreate, onDestroy описують алгоритми дій при запуску, продовженні, зупинці, створенні та знищенні графічного вікна відповідно;
- метод onBackPressed відповідає за виконання дій при натисканні апаратної кнопки «Назад».

Даний клас має дочірній клас UIDataSyncer. Клас UIDataSyncer є нащадком класу AsyncTask (аналог Thread у Android). Даний клас відповідає за

асинхронну взаємодію з фоновими сервісами, якщо вони працюють. Асинхронний алгоритм дозволяє не блокувати графічний інтерфейс користувача та таким чином підвищує позитивний досвід користувача. Поля даного класу зберігають останні отримані дані та статус для відображення. Метод doInBackground є аналогом run у традиційному класі Thread. Метод onProgressUpdate викликається усередині класу або за запитом для виконання у контексті головного потоку.

Клас Algorithm є демоном. Розширює системний клас IntentService та працює у фоновому режимі без прив'язки до головного потоку застосування. Даний алгоритм може знищити операційна система за потребою або власноруч користувач. Реалізує комунікацію між пристроєм, виконує більшість алгоритмів, які пов'язані з смарт-браслетом. Має такі поля:

– поле Database є об'єктом класу DatabaseController, який необхідний для взаємодії з базою даних. Даний клас виконаний та описаний у першої частині даного проекту;
– поле Prefs реалізує доступ до системних налаштувань застосування. У даному випадку використовується для читання MAC-адреси пристрою до якого виконана прив'язка та запису чи читання мінімальних, середніх, максимальних даних серцебиття
– поля ServiceObject, Connection, mBinder системні та необхідні для керування прив'язкою демона до застосування або для його взаємодії з операційною системою;
– поле PhoneListener необхідне для прослуховування апаратного модуля радіозв'язку та відповідно відгуку на події, які з ним пов'язані;
– поле SBIReceiver необхідний для безпосереднього зв'язку з пристроєм. Згідно з мережевою моделлю OSI даний клас розташований найближче до транспортного рівню;

– поля LastMainSync, LastHRSync, LastSleepSync містять дату/час останньої синхронізації загальних даних, серцебиття, даних сну відповідно;
– поле NextSync зберігає дані щодо запланованої наступної синхронізації з пристроєм;
– поле IsRealTimeSynced відображає чи використовується алгоритм синхронізації у реальному часі;
– поле SelfPointer містить покажчик на об'єкт даного класу. Необхідний для безпосередньої взаємодії з демоном. Дане поле статичне;
– поля BatteryHolder, LastHRIncomed, LastStepsIncomed, LastCcalsIncomed, LastStatus зберігають останні отримані дані з смарт-браслету щодо статусу акумулятора, значення серцебиття за п'ять хвилин, кількість пройдених шагів, кількість витрачених калорій відповідно;
– поле MainSyncPeriodSeconds містить значення у секундах, яке використовується як інтервал між синхронізаціями;
– поле ApproachingAlarm містить покажчик на наступний годинник;
– поля IsAlarmWaiting, IsAlarmTriggered, IsAlarmKilled використовуються для визначення чи очікується годинник, чи переведено його у активний стан та чи вимкнув його користувач. Дані поля використовуються у експериментальних алгоритмів для керування усвідомленими сновидіннями.

Даний клас має такі методи:
– GetLastHRSync, GetLastMainSync та GetLastSleepSync відповідають за безпосередньої синхронізації даних серцебиття, другорядних даних та даних сну;
– FillGlobalValues необхідний для аналізу мінімального, максимального та середнього значення серцебиття;
– CheckForAlarmWaiting, CheckForAlarms та AlarmTriggerDecider відповідають за перевірку чи очікується годинник, пошук запланованих

годинників та алгоритми запуску годинника у активний стан. Дані методи використовуються для експериментів з алгоритмами дослідження та керування усвідомлених сновидіннями;
- PostShortMessageDivider використовується у алгоритмів передачі коротких текстових повідомлень до смарт-браслету у рамках даного класу;
- ExecuteForceSync використовується для приведення в дію алгоритму синхронізації;
- ChangeStatus використовується для передачі повідомлень усередині застосування для взаємодії сервісів з графічною частиною застосування;
- onStartCommand, onCreate, Init, onDestroy відповідає за дії при запуску, створенні, ініціалізації та знищення демона відповідно.

Розробка форми відображення збережених даних

Наступною частиною роботи є розробка засобів для відображення отриманих та збережених даних. Виконують дані завдання файли activity_data_view.xml, activity_data_withoutcalendar.xml та DataView.kt. Відмінність варіантів розмітки: перший – використовує графічний календар для можливості групування даних по неділям або місяцям та вивантажувати збережені дані за запитом користувача. Другий варіант розмітки не використовує графічний календар та лише відображує дані у табличному вигляді або у вигляді списку.

Обидва варіанти використовують спільний програмний модуль DataView.kt. Дана форма виконує лише читання з локальної бази даних без можливості модифікації також дана форма використовується не лише для відображення даних синхронізації, а усе що можна представити у табличному вигляді або у вигляді списку. Наприклад перелік встановлених

додатків для яких необхідно застосовувати алгоритми фільтрації повідомлень.

На даній діаграмі зображено, що програмний модуль DataView складається з класів DatabaseController, DataComplexer. Клас DataComplexer складається з класів AsyncTasker та Record.

Клас DataView має такі поля:

- DC – об’єкт класу DatabaseController, який необхідний для взаємодією з фізичною базою даних;
- DHRC – об’єкт класу DataComplexer, у якому закладені алгоритми групування даних;
- LastValue – цілочисельний тип, який необхідний для зберігання останньої обробленої величини;
- CalendarTool – об’єкт класу CalendarPickerView, який необхідний для можливості вибору інтервалу часу, для якого необхідно отримати вибірку даних. Даний клас використовується з зовнішніх бібліотек та не є вбудованим;
- ConfirmBtn – покажчик на кнопку підтвердження вводу. Необхідно для динамічної перебудови вигляду графічного вікна в залежності від типу даних, які необхідно відображати;
- RequestBtn – покажчик на кнопку переключення до вибору інтервалу часу для отримання вибірки з бази даних. Необхідна для динамічної перебудови вигляду графічного вікна в залежності від типу даних для відображення;
- Data – тип даних, закодований у строку, які будуть відображатися;
- MainTable – покажчик на контейнер у якому зберігаються дані для відображення;
- ScaleGroup – покажчик на контейнер у якому знаходяться опції групування даних за інтервалами день, тижні, місяці;
- DayScale – покажчик на кнопку для вибору опції групування даних за дням. Стандартна опція;

– WeekScale – покажчик на кнопку для вибору опції групування даних за тижнями;
– MonthScale – покажчик на кнопку для вибору опції групування даних за місяцями.

Клас DataView складається з таких методів (звичайні системні методи опущено):

– Init – ініціалізація класу, аналог конструктору у мові програмування Kotlin;
– HeaderChoose – містить алгоритми ініціалізації заголовку таблиці при відображені даних, які мають табличний вигляд;
– ExecuteTableHeaderCreation – розміщує згенеровані заголовки таблиці у контейнері MainTable;
– TextViewCreator – генерує примітивну одиницю елементу таблиці чи списку у контейнері MainTable;
– UpdateView – вивантажує дані, які міститься у класі, який групує дані. Необхідний для оптимізації пам’яті та виконання асинхронних алгоритмів вивантаження даних;
– LoadData – завантажує дані до контейнеру MainTable в залежності від того, з яким параметром Data було проініціалізовано даний клас;
– GetLastOrFirstDate – допоміжна функція для роботи з класом CalendarPickerView. Завантажує границі обраного інтервалу користувачем.

Програмний модуль DataView пов’язаний з класом DatabaseController, який відповідає за вивантаження даних з бази даних. Даний клас розроблений та описаний у першої частині цього проекту.

Для групування даних та для асинхронного вивантаження розроблено клас DataComplexer, який складається з класів Record та AsyncTasker. Даний клас є посередником між DataView та AsyncTasker. Клас AsyncTasker у допоміжному потоці генерує об’єкти класу Record, які будуть розташовані у контейнері MainTable.

Клас DataComplexer має такі поля:

- FromLong – закодована дата/час у цілочисельний тип, який є нижньою границею інтервалу;
- ToLong – закодована дата/час у цілочисельний тип, який є верхньою границею інтервалу;
- CurrentLock – поточне значення інтервалу. Необхідно для координування та породження класів AsyncTasker. Даному значення знаходиться у межах ToLong >= CurrentLock >= FromLong;
- StaticOffset – значення кроку інтервалу. Необхідно для виконання групування даних за днями, тижнями, місяцями;
- Mode – закодований у цілочисельний тип режим роботи;
- ViewStyle – закодований у цілочисельний тип режим відображення даних;
- TaskerObject – об’єкт класу AsyncTasker. Необхідний для керування поточним потоком, який генерує об’єкти класу Record;
- Output – контейнер для зберігання згенерованих об’єктів класу Record.
    - Клас DataComplexer містить такі методи:
- SetOffset – встановлює значення шагу інтервалу в залежності від обраного режиму роботи;
- Compute – запускає алгоритм групування даних;
- Init – аналог конструктору у мові програмування Kotlin. Ініціалізує даний клас та задає параметри алгоритму групування та генерації.

Клас AsyncTasker є нащадком класу AsyncTask, що у свою чергу є модифікованою версією класичного Thread. Даний клас генерує об’єкти класів Record.

Клас Record є примітивною одиницею відображення даних при режимі роботи з табличними даними. Містить такі поля:

- When – значення дати/часу або його інтервалу до якого належить значення;

- MainValue – основне значення запису;
- MinValue – значення мінімального значення групи даних. Ініціалізується, якщо DataComplexer працює у режимі групування даних за інтервалами часу;
- MaxValue – значення максимального значення групи даних. Ініціалізується, якщо DataComplexer працює у режимі групування даних за інтервалами часу.

UML-діаграму з цими зв'язками зображено на рис. 3. Буде проілюстровано лише ключові зв'язки першого порядку.

Важливо підкреслити, що клас DataView та графічну розмітку, яку до нього прив'язано можна використовувати для відображення різних табличних типів даних у проекту, а саме від простого типу даних з групою даних та і складних сутностей, які використовують перемикачі чи інші графічні елементи окрім тексту.

Android Studio підтримує завантаження фрагментів для використовування однієї графічної форми для відображення на неї різних графічних елементів. Завантаження іншої форми (Activity) займає деякий час та повністю блокує інтерфейс користувача, що є не дуже практичним при моделюванні графічних інтерфейсів.

З іншого боку до кожного Activity прив'язаний один клас-адаптер, який ним керує. Таким чином при великої кількості фрагментів створюється навантаження на програміста та читабельність коду. З іншого боку перемикання між графічними сутностями відбувається швидше.

Кожен фрагмент описується у відповідному xml файлі та може бути відображений за допомогою виклику команд setContentView або за допомогою статичного класу InflateLayout. Також фрагменти можна викликати у формі діалогів.

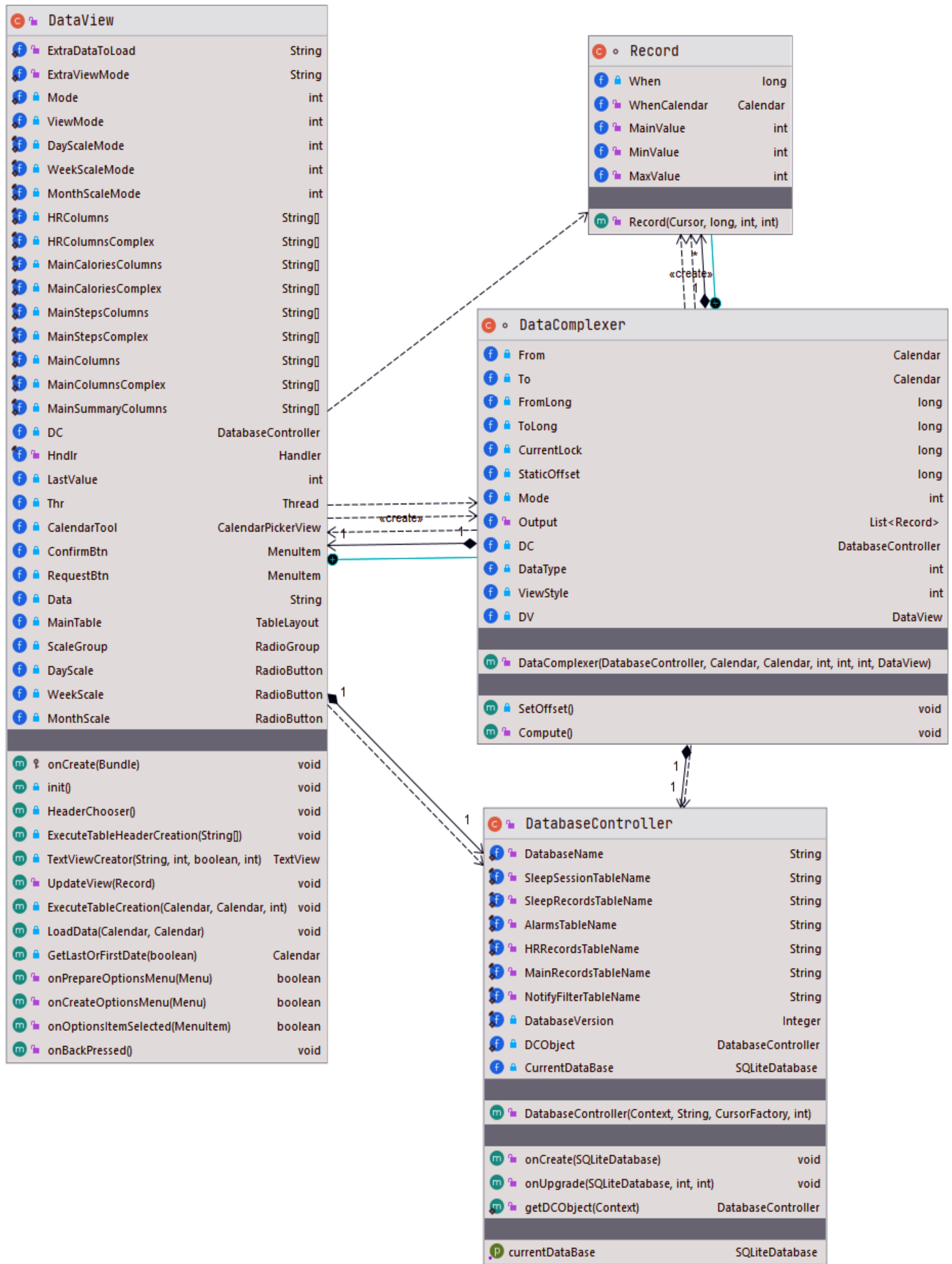

Рисунок 3 – UML-діаграма вікна для відображення даних

Розробка форми керування параметрами застосування

Для керування пороговими значеннями алгоритмів, а також функціями смарт браслету необхідно спроектувати графічну форму для надання можливості користувачу виконувати ці дії. У даному проекті така форма у даному проекті має графічну розмітку у файлі activity_settings.xml, а опис програмного модуля у файлі SettingsActivity.kt.

Даний програмний модуль відображає головні статистичні дані для алгоритмів та передбачає керування алгоритмами фільтрації, відстеження та планування повідомлень або сповіщень до смарт-браслету.

Даний програмний модуль містить внутрішній клас CustomTableRow, який буде використовуватися для відображені у контейнері PackagesList. Даний клас використовує клас Algorithm, який відправляє команди за допомогою генератору команд CommandInterpreter. Даний програмний модуль відрізняється від програмного модуля DeviceControllerActivity тим, що не інстанцює програмну одиницю Alhorithm, а лише використовує її послуги. За допомогою звернень до статичних методів у класі CommandInterpreter генерується послідовність байтів, яка відправляється через програмну одиницю Alhorithm.

У даному програмному модулі можна виконувати керування смарт браслетом, задавати параметри для алгоритмів та вимикати їх при необхідности, а також керувати логікою застосування. На відміну від оригінального застосування даний програмний модуль не має аналога оновлення програмного забезпечення смарт-браслету. Таким чином, якщо розробник смарт-браслету розробить оновлення, то його необхідно буде завантажити через оригінальне застосування.

На рис. 4 зображена UML-діаграма з описом даного програмного модуля та головними зв’язками. Дочірні зв’язки опущено.

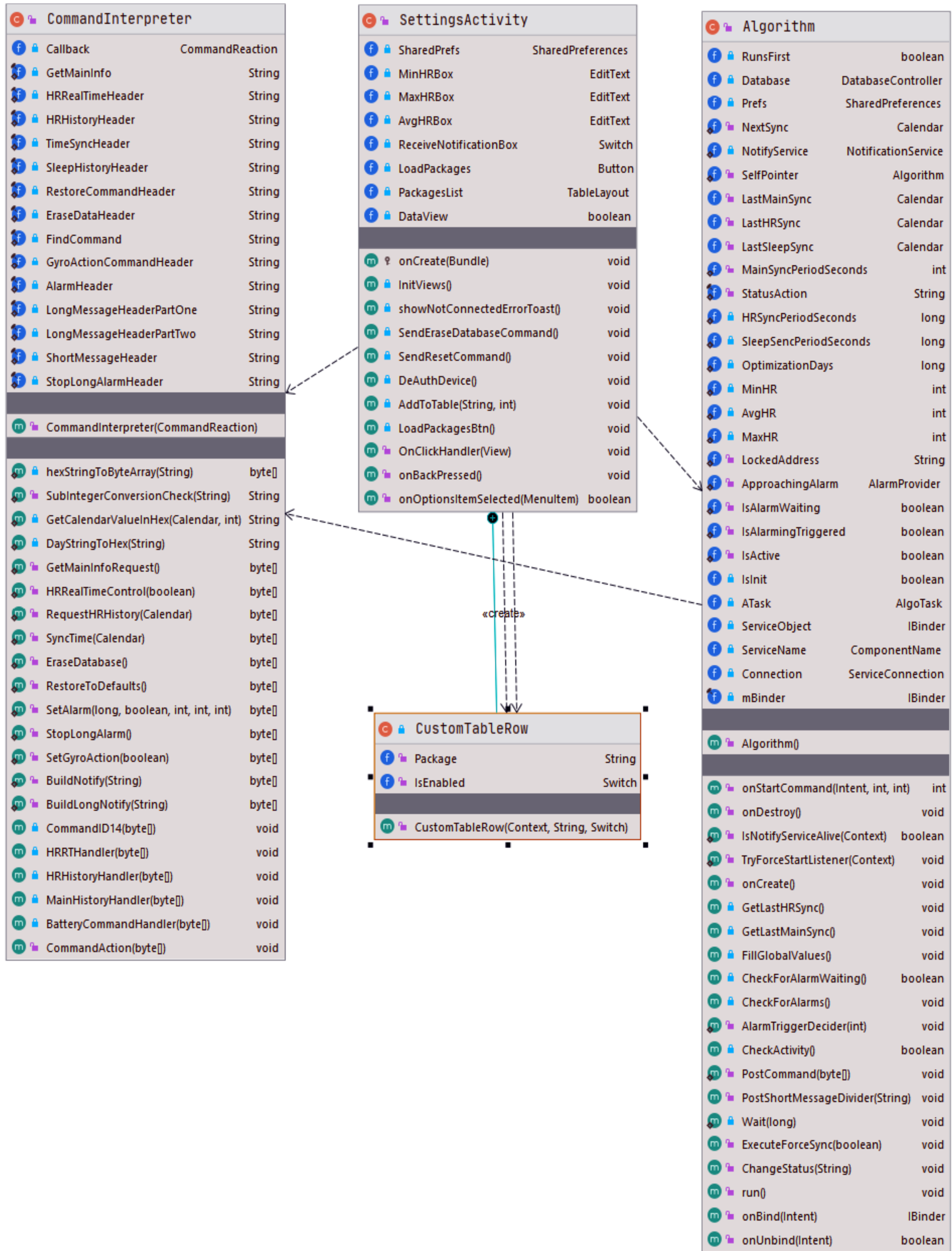

Рисунок 4 – UML-діаграма програмного модуля для додатковим керуванням пристроєм

Клас SettingsActivity має такі поля:

- SharedPrefs – системний клас для запису налаштувань застосування у захищене внутрішнє сховище смартфону.
- MinHRBox – покажчик на текстове поле, яке відображує глобальне мінімальне значення серцебиття за увесь час використання застосування.
- MaxHRBox – покажчик на текстове поле, яке відображує глобальне максимальне значення серцебиття за увесь час використання застосування.
- AvgHRBox – покажчик на текстове поле, яке відображує глобальне середнє значення серцебиття за увесь час використання застосування.
- ReceiveNotificationBox – покажчик на перемикач, який керує увімкненням чи вимкненням алгоритмів пересилання повідомлень.
- ReceivePhoneCallBox – покажчик на перемикач, який керує увімкненням чи вимкненням алгоритмів пересилання статусу дзвінка на смартфон.
- EnableAutoIllumination – покажчик на перемикач, який керує увімкненням чи вимкненням алгоритму автоматичної відображення даних на смарт-браслеті за допомогою вбудованого акселерометра.
- PackagesList – покажчик на контейнер у якому будуть відображуватися примітивні елементи CustomTableRow
- DataView – логічне значення, яке відображує чи знаходиться поточне графічне вікно у режимі відображення списку даних.

У даному класі поля MinHRBox, MaxHRBox, AvgHRBox використовуються для керування експериментальними алгоритмами дослідження керованими сновидіннями.

У даному класі поля ReceiveNotificationBox, ReceivePhoneCallBox керує роботу класу Alhorithm при певних подіях. Наприклад при надходженні повідомлення від певного додатку на смартфоні. При стані «увімкнено» клас,

який прослуховує статус повідомлень на смартфоні буде транслювати повідомлення до смарт-браслету та оброблювати їх за певними алгоритмами. Від стану перемикача у пункті 6 залежить чи буде транслюватися дані дзвінка до смарт-браслету, якщо прослуховувач стану телефону побачить вхідний дзвінок до смартфону.

Клас SettingsActivity має такі методи (стандартні системні методи опущено):

- InitViews – ініціалізує графічні компоненти в залежності від записаних даних налаштувань;
- showNotConnectedErrorToast – відображує повідомлення при помилки з’єднання;
- SendEraseDatabaseCommand – очищує внутрішню пам’ять смарт браслету від записів;
- SendResetCommand – відправляє команду для переходу пристрою до стандартних налаштувань;
- DeAuthDevice – розриває прив’язку до смарт-браслету та переводить застосування до початкового стану. База даних не очищується;
- AddToTable – допоміжний метод, який виконує додавання елементів CustomTableRow до контейнеру PackagesList;
- LoadPackagesBtn – завантажує перелік встановлених додатків на смартфон для керування фільтрацією повідомлень;
- OnClickHandler – метод, який містить опис алгоритмів для кожного елементу на який може натиснути користувач.

Класи Alhorithm та CommandInterpreter описані у розділі 3.2. Клас CustomTableRow є нащадком системного класу TableRow та містить поля для відображення графічної іконки відповідного додатку, назви додатку та перемикач для керування фільтрацією прослуховування повідомлень від даного додатку. Після встановлення перемикача відбувається запис у SharedPrefs (налаштування) та застосування починає оброблювати повідомлення від даного додатку.

При першому запуску налаштувань фільтрації або при увімкнені користувачу буде надісланий запит на розширення прав доступу застосування до повідомлень смартфону. Згідно з політикою безпеки розробниками безпеки користувач повинен у ручному режимі увімкнути дозвіл до обробки повідомлень користувача по запиту застосування. Після позитивної відповіді від користувача застосування зможе оброблювати усі повідомлення у системі та дублювати їх на смарт-браслет.

Розробка форми планування годинників

Смарт-браслет має функцію планування годинників та для можливості їх використовування необхідно розробити програмний модуль. У даному проекті графічна форма даного модуля описана у файлі activity_alarm.xml та програмна логіка у файлі AlarmActivity.

Клас AlarmActivity має внутрішній клас AdvancedTableRow, який розміщується у контейнері Table. Графічно цей клас ілюструє користувачу сутність класу AlarmProvider у текстово-графічному вигляді.

При аналізу пристрою було з’ясовано, що він має обмежену кількість годинників для планування, але за допомогою ресурсів смартфону це обмеження можна минути при наявності постійної синхронізації. При відсутності синхронізації будуть працювати лише ті годинники, які було синхронізовано у останній раз та якщо вони мали статус увімкнених.

Важливо підкреслити, що при відсутності постійної синхронізації не буде можливості здолати це обмеження. Не рекомендується зловживати цією функцію, бо flash пам’ять має обмежену кількість циклів запису.

UML-діаграма даного класу з основними зв’язками відображена на рис. 5.

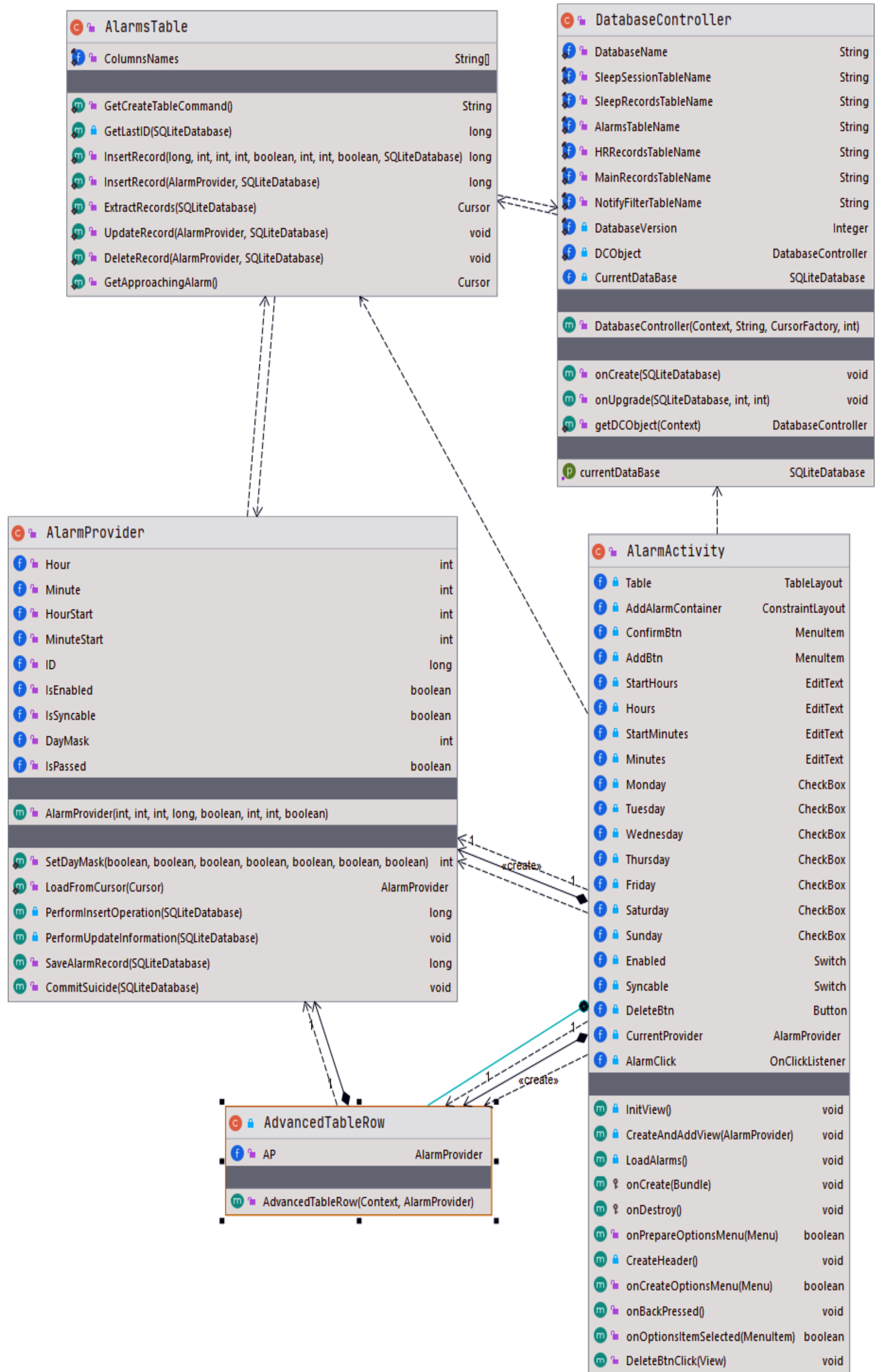

Рисунок 5 – UML-діаграма форми для планування годинників

Клас AlarmActivity має такі поля:

– Table – покажчик на контейнер відображення годинників у графічному вигляді;
– AddAlarmContainer – покажчик на контейнер для розмежування елементів управління від інформації;
– ConfirmBtn – покажчик на кнопку підтвердження додавання годинника. Необхідний для динамічної зміні графічного представлення в залежності від режиму роботи;
– AddBtn – покажчик на кнопку додавання годинника. Необхідний для динамічної зміні графічного представлення в залежності від режиму роботи;
– DeleteBtn – покажчик на кнопку видалення годинника. Необхідний для динамічної зміні графічного представлення в залежності від режиму роботи;
– StartHours – покажчик на текстове поле для вводу значення години, з якої можна привести в дію годинник;
– Hours – покажчик на текстове поле для вводу значення години у яку необхідно привести в дію годинник;
– StartMinutes – покажчик на текстове поле для вводу значення хвилини часу пункту 6 з якої можна привести в дію годинник;
– Minutes – покажчик на текстове поле для вводу значення хвилини часу пункту 7 у яку необхідно привести в дію годинник;
– Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday – покажчики на перемикачі, які відповідають за приведення в дію годинника у дні неділі понеділок, вівторок, середа, четвер, п’ятниця, субота та воскресіння відповідно;
– Enabled – перемикач, який відображує чи увімкнений годинник;
– Syncable – перемикач, який відображує чи потребує даний годинник синхронізацію з смарт браслетом;

– CurrentProvider – містить об’єкт класу над яким у поточний час відбувається взаємодія.

Сутності StartHours, StartHours використовуються у експериментальних алгоритмів для виконання приведення в дію годиннику в залежності від частоти серцебиття користувача. Даний клас дає можливість взаємодії з сутністю AlarmProvider, яка у свою чергу є сутності годиннику з допоміжними алгоритмами.

Клас AlarmActivity має такі методи (системні методи опущено):

– AlarmClick – містить реакцію на відповідь при натисканні на графічну сутність годинника. Перемикає форму у режим редагування обраного годинника;
– InitView – ініціалізує графічну форму, починає завантаження годинників;
– CreateAndAddView – допоміжний метод, який виконує перетворення сутності AlarmProvider у графічне відображення AdvancedTableRow;
– CreateHeader – створює каркас для AdvancedTableRow;
– DeleteBtnClick – описує алгоритм дії при видаленні годинників. Перемикає форму у режим видалення.

У свою чергу AlarmProvider має такі поля і методи:

– поле DayMasks містить значення для побудови бітової маски, які зберігають дні неділі у яких необхідно запускати годинник;
– поле StartHours, Hours, StartMinutes, Minutes IsEnabled, IsSyncable є повним аналогом полей AlarmActivity, а саме пунктів 6, 7, 8, 9, 11, 12 відповідно;
– поле ID зберігає значення ідентифікатору годинника у базі даних;
– поле IsPassed зберігає значення чи було спрацювання годинника;
– метод PerformInsertOperation виконує зберігання нового годинника у базу даних;
– метод PerformUpdateInformation виконує збереження змін внесених до годинника;

– метод SaveAlarmRecord виконує збереження годинника, а саме обирає необхідно створити новий запис чи існуючий;
– метод CommitSuicide виконує видалення годинника з бази даних;
– статичний метод IsAlarmsEqual виконує порівняння годинників. Необхідний для перевірок перед плануванням;
– статичний метод SetDayMask повертає значення бітової маски відповідно до обраних днів;
– статичний метод LoadFromCursor надає взаємодію для ініціалізації об’єкту класу AlarmProvider.

4 РОЗРОБКА ДОПОМІЖНИХ МОДУЛЕЙ

Було розроблено та реалізовано програмні модулі, які розширюють чи реалізують функції смарт-браслету, але необхідно розробити програмні модулі, які реалізують та використовують функції операційної системи Android.

SDK та ANDK, які надаються компанією Google містять усі необхідні засоби для прослуховування стану радіомодулю [9][1)], який реагує на події вхідного, вихідного дзвінка та його завершення. Іншою основним інструментом є засоби для прослуховування повідомлень від інших додатків [10][2)], а також засіб для виконання програмного коду після завантаження чи перезавантаження операційної системи. Будуть використані лише засоби для прослуховування стану радіомодуля та прослуховування повідомлень.

Розробка програмного модуля прослуховування станів

У даному проекті модуль прослуховування описаний у файлі NotificationService.kt.

Даний клас є нащадком класу NotificationListenerService та перезавантажує методи оригінального класу. Даний клас схожий на клас IntentService/Service та виконується у окремому потоці. UML-діаграму з основними зв’язками даного класу наведено на рис. 6.

Важливо підкреслити, що для запуску даного модуля користувач повинен у ручному режимі у налаштуваннях операційної системи дозволити застосуванню отримувати доступ до сповіщень у системі.

[1)] [9] PhoneStateListener | Android Developers URL:https://developer.android.com/reference/android/telephony/PhoneStateListener. (дата звернення 15.03.2020)

[2)] [10] NotificationListenerService | Android Developers. URL:https://developer.android.com/reference/android/service/notification/ NotificationListenerService. (дата звернення 15.03.2020)

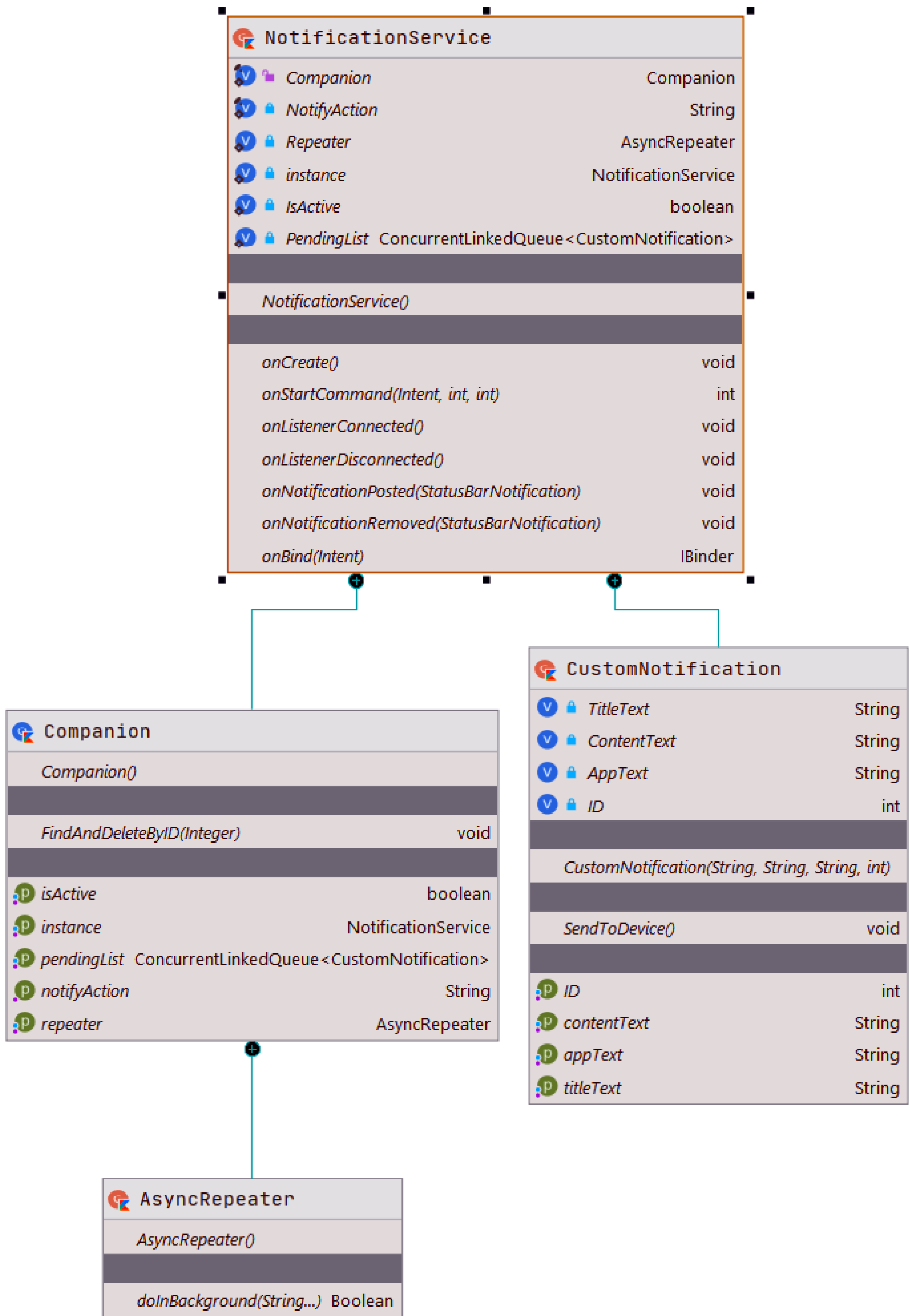

Рисунок 6 – UML-діаграма модуля прослуховування сповіщень

Даний модуль складається з допоміжного класу CustomNotification, яке є сутністю, яке дублює логіку та властивості сповіщень у програмному коді. Має такі поля та методи:

- поле AppText – назва пакету чи додатку, яке згенерувало повідомлення;
- поле TitleText – заголовок повідомлення;
- поле ContextText – вміст повідомлення;
- поле ID – внутрішній ідентифікатор повідомлення, яке генерується операційною системою Android;
- метод init є конструктором;
- метод SendToDevice виконує відправку об’єкту класу CustomNotification до смарт-браслету.

Клас AsyncRepeater реалізує додатковий алгоритм, а саме повторення надсилання повідомлень, які не були переглянуті або відмінені на самому смартфоні. Даний клас є нащадком класу AsyncTask, який є розвинутим аналогом класичного Thread. Даний клас виконується у іншому потоці та діє незалежно.

Клас NotificationService має такі методи та поля (стандартні методи опущені):

- статичне поле Repeater містить посилання на об’єкт класу AsyncRepeater для безпосереднього керування потоком додаткового алгоритму;
- статичне поле instance містить посилання на об’єкт класу NotificationService для безпосереднього керування модулем прослуховування повідомлень;
- статичне поле IsActive керує режимом роботи модуля NotificationService;
- статичне поле PendingList є асинхронним списком, яке містить повідомлення, які необхідно оброблювати;

– метод onListenerConnected системний метод, який викликається при приєднанні модуля до операційної системи. Викликається після того як користувач дозволив застосуванню оброблювати внутрішні повідомлення смартфону;
– метод onListenerDisconnected системний метод, який викликається якщо користувач вручну видалив доступ застосування оброблювати системні повідомлення;
– метод onNotificationPosted викликається після того як зовнішній додаток згенерував повідомлення;
– метод onNotificationRemoved викликається після того користувач видалив повідомлення або воно було зачинено чи видалено додатком, яке його згенерувало;
– метод FindAndDeleteByID видаляє повідомлення з списку повідомлень, які оброблюються PendingList.

Програмний модуль для прослуховування стану радіомодуля у даному проекті описано у файлі PhoneStateListenerBroadcast.kt. У даному проекті було використано замість класу PhoneStateListener клас BroadcastReceiver. Можливостей даного класу досить для виконання усіх необхідних функцій. Розширення та використання класу PhoneStateListener має забагато зайвих функцій використання яких у даному проекті непередбачено.

Даний програмний модуль є нащадком класу BroadcastReceiver та перезавантажує єдиний метод onReceive. У файлі маніфесту даний прослуховувач запрограмований на фільтрацію повідомлень від радіомодуля. Радіомодуль має три стану: дзвінок, звичайний стан, відключення від дзвінка. При дзвінку дані дзвінка транслюється на смарт-браслет за допомогою команд LongMessageHeaderPartOne та LongMessageHeaderPartTwo. Якщо дзвінок прийнято або скинуто, то повідомлення до смарт-браслету видаляється за допомогою команди StopLongAlarmHeader.

Розробка програмного модуля для передачі повідомлень

Для передачі повідомлень до смарт-браслету необхідно використовувати програмний адаптер Bluetooth. Згідно з документації з першої частини проекту для зв’язку використовується інтерфейс UART, який було успішно проаналізовано програмним засобом Wireshark.

Для написання даного модуля за каркас було використано бібліотеку Android-nRF-UART [11][1)]. Дана бібліотека має вже готовий набір функцій для простої взаємодії з BLE-пристроями, а також за замовчуванням має перелік ідентифікаторів для розпізнавання кожного сервісу у BLE-пристрою.

Важливо підкреслити, що у кожного BLE пристрою може бути декілька сервісів або характеристик. Наприклад, характеристика з властивістю лише читання, для отримання серійного номеру пристрою. У даному класі ідентифікатор UART сервісу зберігається в полі RX_CHAR_UUID для читання послідовних даних та TX_CHAR_UUID для запису послідовних даних до пристрою. При необхідності можна власноруч замінити ці константи, якщо виробник пристрою використовує інші ідентифікатори. Очевидно, що у випадку, якщо пристрій не використовує послідовний алгоритм передачі даних, але ідентифікатори сервісів збігаються, то комунікацію не буде встановлено. Повинні збігатися як алгоритми так і ідентифікатори.

У даному класі використовуються лише послуги для послідовної комунікації з приладом та перевірка рівню заряду акумулятора пристрою. Інші приклади, наприклад ідентифікатор для виконування оновлення програмного забезпечення не використовуються або не були проаналізовані.

UML-діаграма даного програмного модуля проілюстрована на рис. 7.

[1)] [11] GitHub - NordicPlayground/Android-nRF-UART: nRF UART app for Android. A simple app showing how to handle BLE with custom service in Android. URL: https://github.com/NordicPlayground/Android-nRF-UART. (дата звернення 17.03.2020)

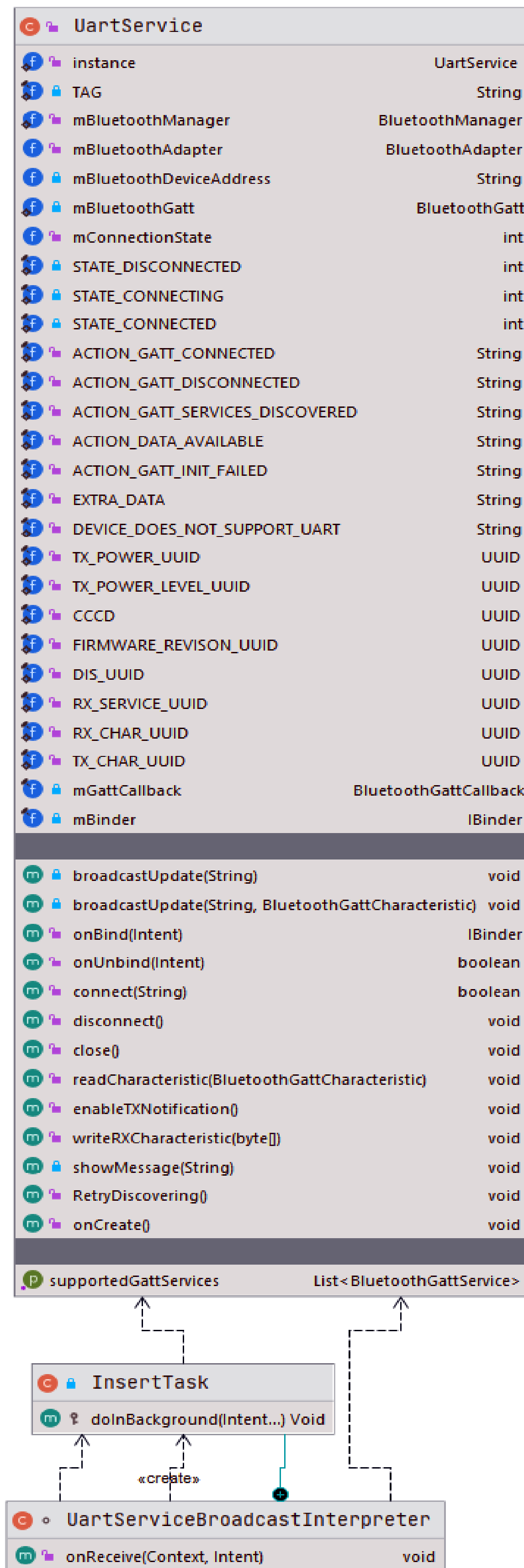

Рисунок 7 – UML-діаграма програмного адаптеру Bluetooth

На даній діаграмі зображений клас UartService який є нащадком класу Service. Виконується незалежно від головного потоку та використовуються для асинхронної взаємодії застосування з апаратним забезпеченням Bluetooth. Повідомлення можуть приходити та відправлятись у будь який час тому необхідна підтримка асинхронних алгоритмів. Дану підтримку забезпечує клас UartServiceBroadcastInterpter, який отримує повідомлення за допомогою внутрішньої системи комунікації Android.

Клас InsertTask виконує асинхронне додавання даних до бази даних або внесення їх до застосування, якщо воно активно. Усі операції взаємодії з базою даних або тривалі за часом операції необхідно розробити з підтримкою асинхронності для того, щоб виключити блокування інтерфейсу, що більшість користувачів знаходить як дратующе. У Android для того щоб підтримувати асинхронність необхідно створити клас AsyncTask та у тіло doInBackground помістити усі тривалі операції. Даний клас має можливості оновлювати інтерфейс, якщо перевантажити метод onProgress.

Клас UartService має такі методи та поля (системні методи опущено):

– поле mBluetoothDeviceAddress містить MAC-адресу пристрою з яким відбувається комунікація;
– поле mBluetoothGatt є системним класом, який є сутністю окремого атрибуту BLE пристрою;
– поле mBluetoothGattCallback описує реакцію на події при взаємодією з атрибутами BLE пристрою, а саме на події зміни статусу з’єднання, сканування та знаходження GATT характеристики пристрою, читання характеристики GATT, зміна стану характеристики GATT;
– поле mBluetoothAdapter містить посилання на об’єкт класу, який виконує безпосереднє керування технологією Bluetooth;
– поле mConnectionState містить закодоване значення статусу з’єднання у цілочисельний тип;
– статичне поле instance містить посилання на об’єкт класу UartService для безпосереднього керування інтерфейсом UART;

- поле mBluetoothManager містить посилання на системний клас, яке керує статусом технології Bluetooth за допомогою операційної системи Android;
- поля STATE_CONNECTED, STATE_DISCONNECTED та STATE_CONNECTING зберігають константи цілих цифр, для ідентифікації стану бездротового з’єднання;
- поля ACTION_GATT_CONNECTED, ACTION_GATT_DISCONNECTED, ACTION_GATT_SERVICES_DISCOVERED, ACTION_DATA_AVALIABLE, ACTION_GATT_INIT_FAILED та EXTRA_DATA зберігаються строкові константи для ідентифікації стану програмного модуля. Дані константи також транслюються через внутрішню систему сповіщень для обробки програмним модулем UartServiceBroadcastInterpreter
- поля TX_POWER_UUID та TX_POWER_LEVEL_UUID зберігають константи ідентифікаторів для сервісів, які записують у канал зв’язку стан заряду акумулятора пристрою;
- методи broadcastUpdate виконують передачу повідомлення у внутрішній системі комунікації Android, яке буде оброблено класом UartServiceBroadcastInterpreter;
- метод showMessage використовується для тестування;
- метод writeRXCharacteristic записує байтові послідовність до атрибуту GATT, яка запрограмована для роботи згідно з інтерфейсом UART;
- метод enableTXCharacteristic виконує увімкнення атрибуту GATT, яке використовується для виконання певної дії. У даному інтерфейсі це аналог натискання кнопки;
- метод RetryDiscovering запускає послідовність пошуку GATT атрибутів;

– методи close, disconnect, connect виконують закриття інтерфейсу зв'язку Bluetooth, відключення від фізичного пристрою та підключення до фізичного пристрою відповідно.

Розробка маніфестного модуля

Після завершення розробки застосування стає відомо які будуть використовуватися функції та додаткові запити до операційного системи, які необхідно внести до файлу AndroidManifest.xml.

Інша частина даного файлу містить описи приймачів внутрішніх повідомлень від операційної системи. У файлі маніфесту запитуються такі функції операційної системи:

```
    <uses-permission
android:name="android.permission.BLUETOOTH" />
    <uses-permission
android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission
android:name="android.permission.BIND_NOTIFICATION_LISTENER_SERV
ICE" tools:ignore="ProtectedPermissions" />
    <uses-permission
android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission
android:name="android.permission.READ_CONTACTS" />
    <uses-permission
android:name="android.permission.READ_PHONE_NUMBERS" />
    <uses-permission
android:name="android.permission.WAKE_LOCK"/>
```

Таким чином запитуються права доступу до технології Bluetooth та ручним його керуванням. Запит до функції Bluetooth Admin дає можливість керувати з'єднаннями по інтерфейсу Bluetooth без участі користувача.

Запит функції ACCESS_COARSE_LOCATION запитує доступ до неточного виявлення місце-знаходження. Згідно з документації Android [12][1] необхідно зробити запит до даної функції для смартфонів з версією API 28 та вище або ACCESS_FINE_LOCATION для версій нижче 28. Необхідний для пошуку пристроїв BLE. При ігноруванні даного запиту увесь функціонал застосування не буде працювати.

Запит функції READ_EXTERNAL_STORAGE та WRITE_EXTERNAL_STORAGE надає дозвіл застосуванню для запису на зовнішній носій інформації (наприклад SD-карту). Це необхідно для надання користувачу безпосереднього доступу до фізичної бази даних. При звичайних обставинах (запис до внутрішнього сховища) користувач може втратити дані, якщо пристрій буде скинуто та не має можливості у любий час отримати доступ до фізичною базою даних. При ігноруванні або відхиленні даного запиту базу даних буде збережено у закритий простір застосування. Згідно з документацією [13][2] починаючи з API 23 існують функції які необхідно у ручному режимі запитати у користувача. Дані дві функції входять у цей список.

Запит android.permission.BIND_NOTIFICATION_LISTENER_SERVICE запитує дозвіл до обробки повідомлень від зовнішніх додатків, але потребує додаткового запиту у користувача. При відхиленні або ігноруванні функціонал, який пов’язаний з транслюванням сповіщень не буде працювати.

Запит READ_PHONE_STATE надає можливість застосуванню отримувати доступ до стану радіомодулю. Потребує додатковий запит до користувача починаючи з версії Android 7.0. При відхиленні запиту застосування не буде сповіщати користувача про вхідні дзвінки.

Запит READ_CONTACTS та READ_PHONE_NUMBERS надає можливість переглядати список абонентів на смартфоні. Необхідний для виводу імені абоненту на пристрій або номер телефону на смарт-браслет. Якщо

[1] [12] BroadcastReceiver | Android Developers URL: https://developer.android.com/reference/android/content/BroadcastReceiver. (дата звернення 20.03.2020)

[2] [13] Request App Permissions | Android Developers. URL: https://developer.android.com/training/permissions/requesting. (дата звернення 20.03.2020)

користувач відхилює запит READ_CONTACTS, то функціонал пов'язаний з відображенням іменем абонента або його номеру не буде працювати, але усі інші функції застосування будуть продовжувати функціонувати.

Також у даному файлі маніфесту міститься запит апаратних засобу Bluetooth Low Energy. Застосування не буде встановлено на смартфонах, які не мають даного апаратного засобу. Також при публікації застосування до платформи Google Play дане застосування буде автоматично вилучено з результатів пошуку на тих пристроях, які не підтримують технологію BLE.

Усі запити, які оброблюються користувачем можна у ручному режимі відкликати у операційній системі Android у налаштуваннях відповідного застосування. «Складні запити» такі як прослуховувачі сповіщень необхідно відкликати у налаштуваннях Android «Адміністратори пристрою» або «Агенти пристрою».

ВИСНОВКИ

В ході роботи було проведено аналіз предметної області, обрано актуальні технології для розробки та моделювання програмного забезпечення. Було отримано документацію та інструкції [14][1)] для розробки програмного забезпечення для керування смарт-браслетом. Командна розробка проекту дозволило покращити часові характеристики розробки проекту та розділити проект на незалежні частини.

Було досліджено та вивчено документацію Android SDK та Android NDK, яке було необхідно для розробки програмного застосування, яке згодом було протестовано та налагоджено.

Ураховуючи обсяг проекту та предметну область є можливості до розвинення проекту, а також шляхи та стратегії розвитку які передбачають покращення графічного дизайну програмного застосування, а також її алгоритмічної складової.

Модульна складова даного проекту дозволяє працювати з любими смарт-браслетами при наявності командного модуля та схожими по структурі типами даних та алгоритмів.

[1)] [14] Тюртюбек Є. М. Підсистема управління інтерфейсом зв'язку і базою даних: Бакалаврська кваліфікаційна робота. – ОДЄКУ, Одеса, 2020 рік. – С. 32-41

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Bluetooth low energy overview | Android Developers. URL: https:// developer.android.com/guide/topics/connectivity/bluetooth-le. (дата звернення 03.03.2020)
2. Історія версій Android – Вікіпедія. URL: https://ru.wikipedia.org/wiki/ %D0%98%D1%81%D1%82%D0%BE%D1%80%D0%B8%D1%8F_ %D0%B2%D0%B5%D1%80%D1%81%D0%B8%D0%B9_Android. (дата звернення 03.03.2020)
3. Android Studio – Вікіпедія. URL:https://ru.wikipedia.org/wiki/ Android_Studio. (дата звернення 05.03.2020)
4. MVP – Вікіпедія. URL: https://ru.wikipedia.org/wiki/Model-View-Presenter. (дата звернення 06.03.2020)
5. Github – Вікіпедія URL: https://ru.wikipedia.org/wiki/GitHub. (дата звернення 07.03.2020)
6. Джейсон Мак-Колм Смит. Елементарні шаблони проектування = Elemental Design Patterns. – М.: «Вильямс», 2012. – 304 с. – ISBN 978-5-8459-1818-5.
7. Мартин Фаулер. Шаблони корпоративних застосувань. М.: «Вильямс», 2012. – 544 с. – ISBN 978-5-8459-1611-2
8. OMG | Object Management Group. URL: https://www.omg.org. (дата звернення 10.08.2019).
9. PhoneStateListener | Android Developers. URL: https:// developer.android.com/reference/android/telephony/PhoneStateListener. (дата звернення 15.01.2020)
10. NotificationListenerService | Android Developers. URL:https:// developer.android.com/reference/android/service/notification/ NotificationListenerService. (дата звернення 15.01.2020)
11. GitHub - NordicPlayground/Android-nRF-UART: nRF UART app for Android. A simple app showing how to handle BLE with custom service

in Android. URL: https://github.com/NordicPlayground/Android-nRF-UART. (дата звернення 15.01.2020)

12. BroadcastReceiver | Android Developers. URL:https:// developer.android.com/reference/android/content/BroadcastReceiver. (дата звернення 15.01.2020)
13. Request App Permissions | Android Developers. URL: https:// developer.android.com/training/permissions/requesting. (дата звернення 15.01.2020)
14. Тюртюбек Є. М. Підсистема управління інтерфейсом зв'язку і базою даних: Бакалаврська кваліфікаційна робота. – ОДЄКУ, Одеса, 2020 рік. – С. 32-41

Додаток А

Представлення основних графічних форм

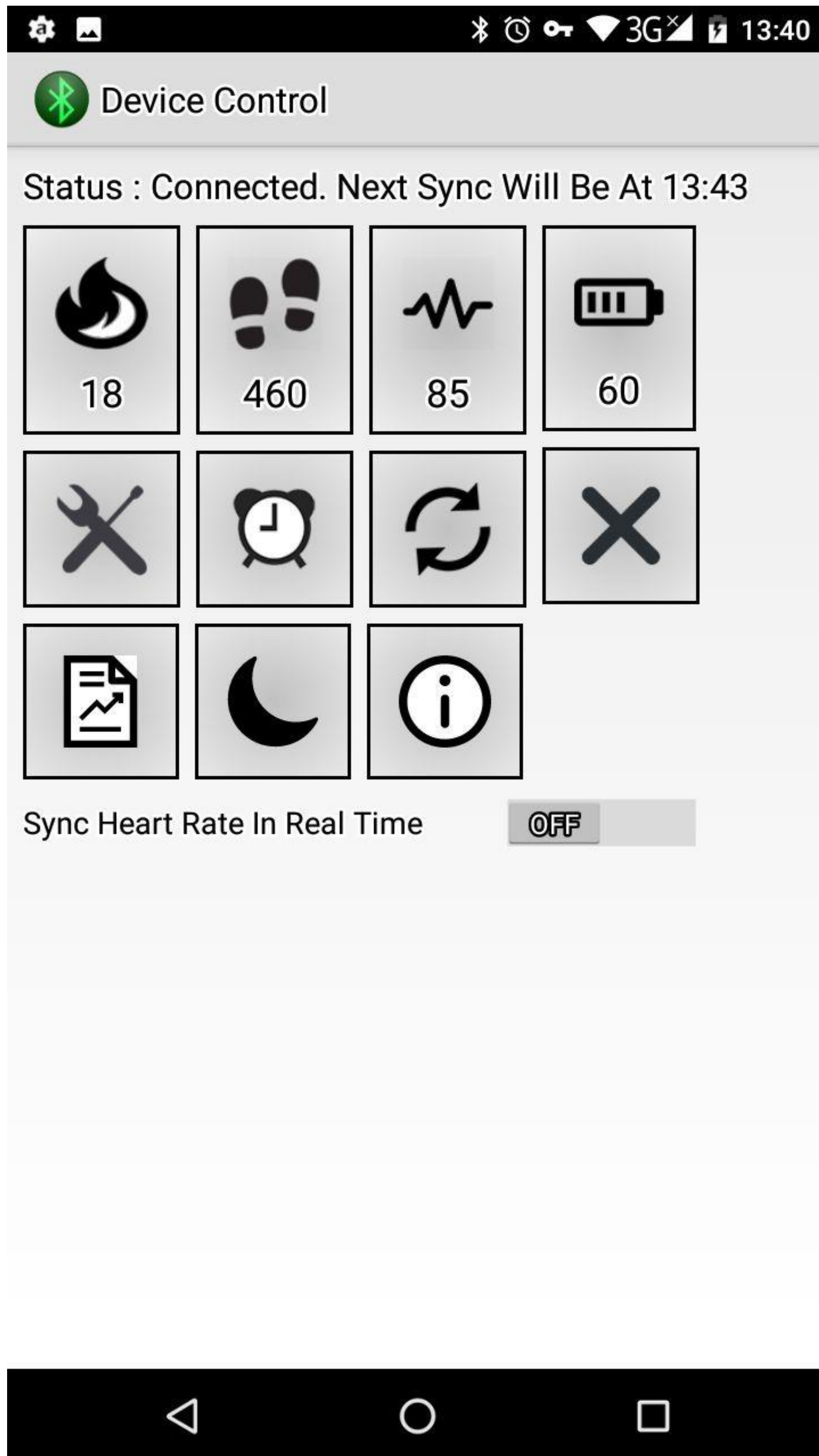

Рисунок 8 – Головна форма керування пристроєм

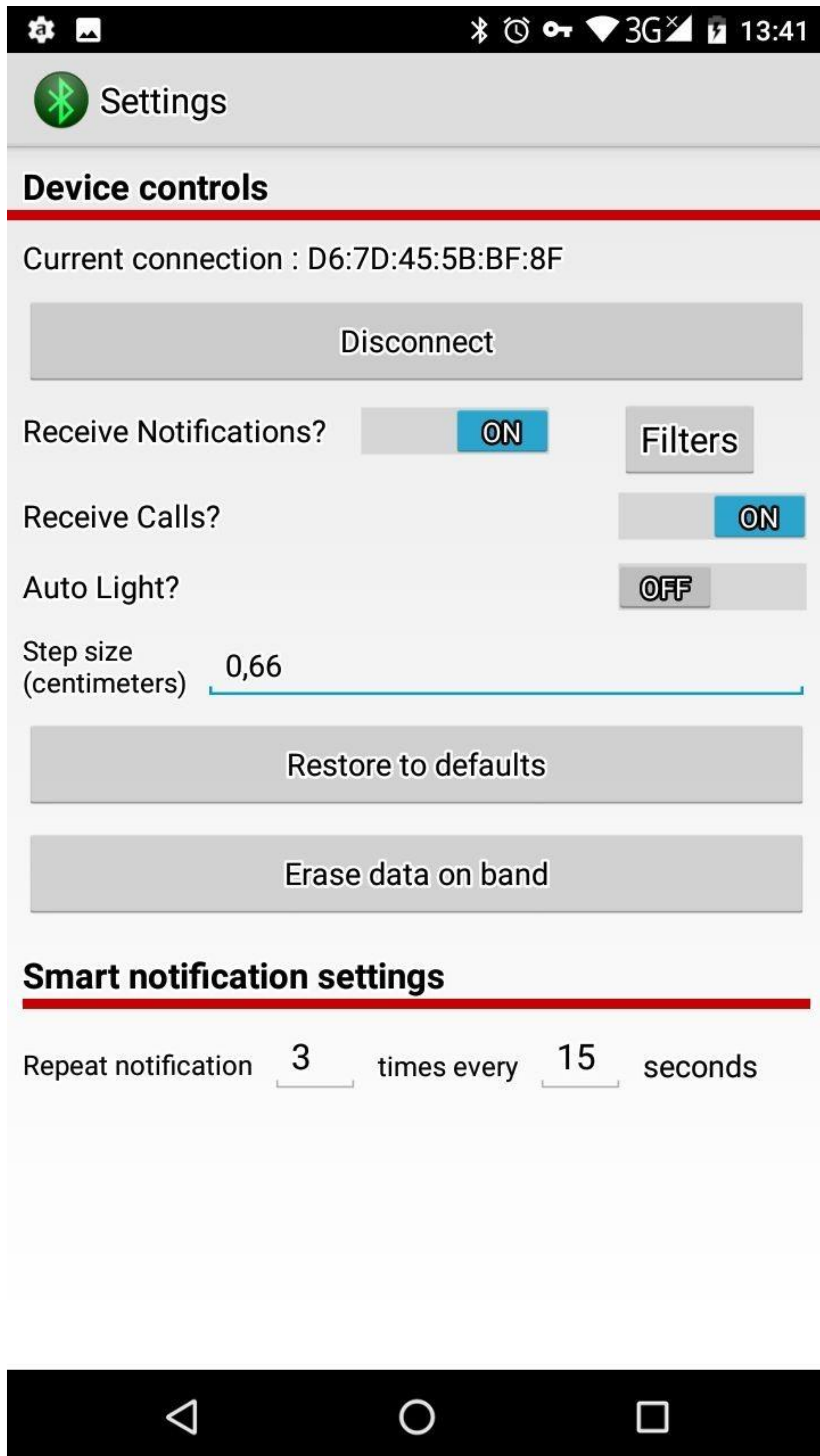

Рисунок 9 – Вікно налаштувань застосування

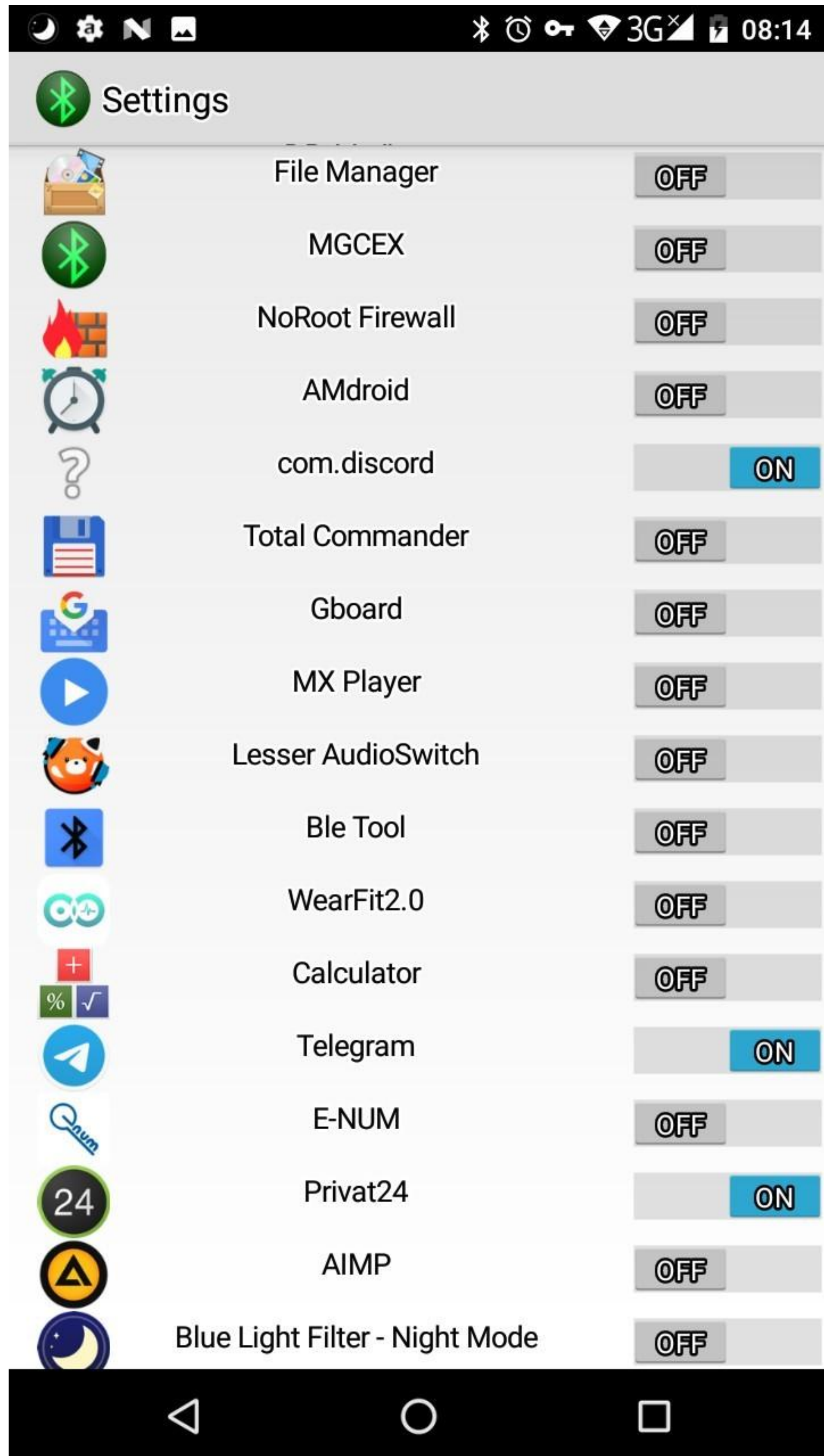

Рисунок 10 – Вікно налаштувань сповіщень

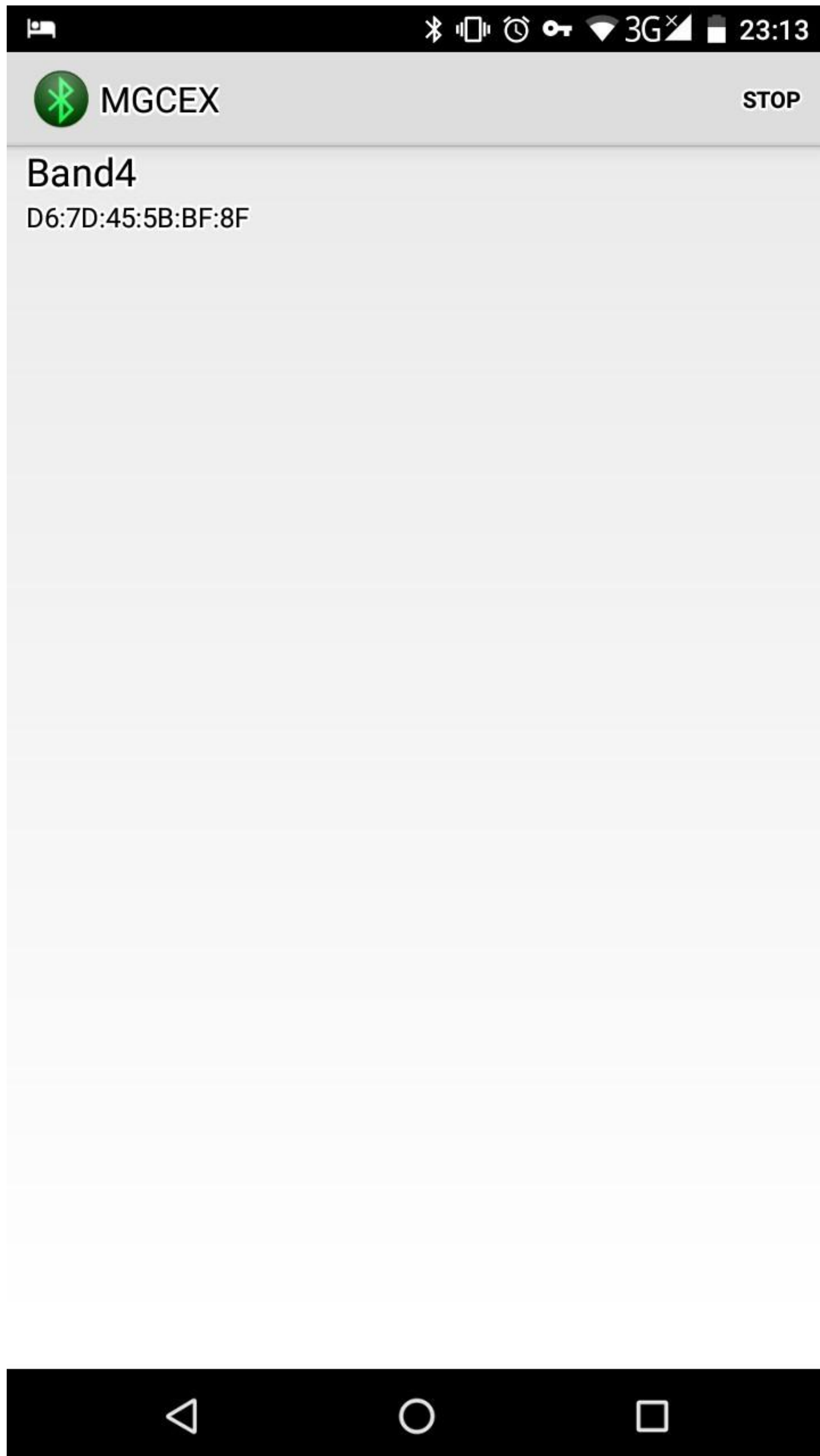

Рисунок 11 – Вікно пошуку пристрою

Рисунок 12 – Вибір типу групування даних та інтервалу вивантаження даних

Data CHOOSE PERIOD

Date	Min	Average	Maximum
September 2019	46	74	183
November 2019	46	71	176
December 2019	46	70	174
February 2020	47	81	168
March 2020	46	67	136

Data CHOOSE PERIOD

Date	Min	Average	Maximum
January 4 2020	46	69	140
January 5 2020	46	72	140
February 2 2020	49	63	86
February 4 2020	58	96	168
February 5 2020	46	66	119

Data CHOOSE PERIOD

Date\Time	Value
May 16 20:00	76
May 16 20:05	68
May 16 20:10	67
May 16 20:15	70
May 16 20:20	64
May 16 20:25	83
May 16 20:30	56
May 16 20:35	60
May 16 20:40	67
May 16 20:45	93
May 16 20:50	83
May 16 20:55	67
May 16 21:00	93
May 16 21:10	95
May 16 21:15	94
May 16 21:20	92
May 16 21:25	75
May 16 21:30	83
May 16 21:35	86

Рисунок 13 – Відображення даних серцебиття згрупованих по місяцям, тижням та без групування відповідно

Додаток Б

Лістинг програмного коду алгоритму синхронізації

```
package anonymouls.dev.MGCEX.App

import android.annotation.SuppressLint
import android.app.*
import android.content.ComponentName
import android.content.Context
import android.content.Intent
import android.content.IntentFilter
import android.content.ServiceConnection
import android.content.SharedPreferences
import android.content.pm.PackageManager
import android.os.*
import android.support.v4.app.NotificationManagerCompat

import java.text.SimpleDateFormat

import anonymouls.dev.MGCEX.DatabaseProvider.AlarmsTable
import anonymouls.dev.MGCEX.DatabaseProvider.CustomDatabaseUtils
import anonymouls.dev.MGCEX.DatabaseProvider.DatabaseController
import anonymouls.dev.MGCEX.DatabaseProvider.HRRecordsTable
import anonymouls.dev.MGCEX.DatabaseProvider.MainRecordsTable
import anonymouls.dev.MGCEX.util.HRAnalyzer
import anonymouls.dev.MGCEX.util.Utils
import java.util.*

class Algorithm : IntentService("Syncer") {
    override fun onHandleIntent(intent: Intent?) {
        Thread.currentThread().name     = "Syncer"
        Thread.currentThread().priority = Thread.MIN_PRIORITY
        run()
    }

    private var Database: DatabaseController? = null
    private var Prefs: SharedPreferences? = null

    var LastMainSync = Calendar.getInstance()
    var LastHRSync = Calendar.getInstance()
    var LastSleepSync = Calendar.getInstance()
    private var ServiceObject: IBinder? = null
    private var ServiceName: ComponentName? = null
    private var hardTask: AsyncTask<Void, Void, Void>? = null

    private val Connection = object : ServiceConnection {
        override fun onServiceConnected(name: ComponentName,
service: IBinder) {
            ServiceObject = service
```

```
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.N)
{
                ServiceName = name
                TryForceStartListener(applicationContext)
            }
        }
        override fun onServiceDisconnected(name: ComponentName)
{}
    }
    private val mBinder: IBinder? = null

    override fun stopService(name: Intent?): Boolean {
        IsActive = false
        return super.stopService(name)
    }
    @SuppressLint("WrongConstant")
    override fun onStartCommand(intent: Intent, flags: Int,
startId: Int): Int {
        return super.onStartCommand(intent,
Service.START_STICKY, startId)
    }
    fun Init(){
        if (IsInit) return
        val DCAct = Intent(this,
DeviceControllerActivity::class.java)
        DCAct.flags = Intent.FLAG_ACTIVITY_CLEAR_TOP or
Intent.FLAG_ACTIVITY_SINGLE_TOP
        var Service = Intent(this, UartService::class.java)
        startService(Service)
        Service = Intent(this, NotificationService::class.java)
        if (!NotificationService.IsActive) {
            bindService(Service, Connection,
Context.BIND_AUTO_CREATE)
            NotificationService.IsActive = true;
        }
        if (!IsNotifyServiceAlive(this))
            TryForceStartListener(this)
        LockedAddress =
Utils.GetSharedPrefs(this).getString("BandAddress", null)
        Prefs = Utils.GetSharedPrefs(this)
        PhoneListener = PhoneStateListenerBroadcast()
        SBIReceiver = UartServiceBroadcastInterpreter()
        registerReceiver(SBIReceiver,
DeviceControllerActivity.makeGattUpdateIntentFilter())
        val IF = IntentFilter("PHONE_STATE")
        if
(Utils.GetSharedPrefs(this).getBoolean("ReceiveCalls", true))
            registerReceiver(PhoneListener, IF)
        IsInit = true
        SelfPointer = this
        Database = DatabaseController.getDCObject(this)
        GetLastHRSync()
```

```
        GetLastMainSync()
    }

    override fun onDestroy() {
        unregisterReceiver(SBIReceiver)
        unregisterReceiver(PhoneListener)
        super.onDestroy()
    }
    override fun onCreate() {
        super.onCreate()
        Init()
    }
    private fun GetLastHRSync() {
        if (Database != null)
            LastHRSync =
CustomDatabaseUtils.GetLastSyncFromTable(DatabaseController.HRRe
cordsTableName,
                    HRRecordsTable.ColumnsNames, true,
Database!!.currentDataBase!!)
    }
    private fun GetLastMainSync() {
        if (Database != null)
            LastMainSync =
CustomDatabaseUtils.GetLastSyncFromTable(DatabaseController.Main
RecordsTableName,
                    MainRecordsTable.ColumnNames, true,
Database!!.currentDataBase!!)
    }
    private fun CheckForAlarmWaiting(): Boolean {
        val Now = Calendar.getInstance()
        val NowHour = Now.get(Calendar.HOUR_OF_DAY)
        val NowMinute = Now.get(Calendar.MINUTE)
        if (ApproachingAlarm!!.HourStart <= NowHour ||
ApproachingAlarm!!.HourStart == NowHour
                && ApproachingAlarm!!.MinuteStart <= NowMinute)
{
            IsAlarmWaiting = true
        }
        return IsAlarmWaiting
    }
    private fun CheckForAlarms() {
        val Alarm = AlarmsTable.GetApproachingAlarm()
        if (Alarm.count > 0) {
            do {
                val Buff = AlarmProvider.LoadFromCursor(Alarm)
                if
(AlarmProvider.IsAlarmsEqual(ApproachingAlarm, Buff)) {
                    if (!IsAlarmKilled) CheckForAlarmWaiting()
                } else {
                    ApproachingAlarm =
AlarmProvider.LoadFromCursor(Alarm)
                    IsAlarmKilled = false
```

```
                }

            } while (Alarm.moveToNext())
        }
    }
    fun AlarmTriggerDecider(HRValue: Int) {
        val Hour = Calendar.getInstance().get(Calendar.HOUR)
        val Minute = Calendar.getInstance().get(Calendar.MINUTE)
        if (IsAlarmKilled) {
            AlarmFiredIterator = 0
            return
        }
        if (HRValue >= AvgHR || IsAlarmingTriggered
                || Hour >= ApproachingAlarm!!.Hour && Minute >=
ApproachingAlarm!!.Minute) {
            IsAlarmingTriggered = true
            if (ApproachingAlarm!!.DayMask == 128)
ApproachingAlarm!!.IsEnabled = false
            PostCommand(CommandInterpreter.BuildLongNotify("WAKE
UP!"), true)
            val resultIntent = Intent(this,
UartServiceBroadcastInterpreter::class.java)
            resultIntent.action = "AlarmAction"
            val resultPendingIntent =
PendingIntent.getBroadcast(this, 0, resultIntent,
PendingIntent.FLAG_UPDATE_CURRENT)
            val builder = if (Build.VERSION.SDK_INT >=
Build.VERSION_CODES.O) {
                Notification.Builder(this, "ALARMS")
                        .setSmallIcon(R.drawable.ic_launcher)
                        .setContentText("Maybe you should kill
it?")
                        .setContentTitle("Alarm is ringing")
                        .setContentIntent(resultPendingIntent)
            } else {
                Notification.Builder(this)
                        .setSmallIcon(R.drawable.ic_launcher)
                        .setContentText("Maybe you should kill
it?")
                        .setContentTitle("Alarm is ringing")
                        .setContentIntent(resultPendingIntent)
            }
            val notification = builder.build()
            val notificationManager =
getSystemService(Context.NOTIFICATION_SERVICE) as
NotificationManager
            notificationManager.notify(21, notification)
        }
    }

    fun PostShortMessageDivider(Input: String) {
        val Req = CommandInterpreter.BuildNotify(Input)
```

```
        var Req1 = Arrays.copyOfRange(Req, 0, 20)
        PostCommand(Req1, false)
        Thread.sleep(50)
        Req1 = Arrays.copyOfRange(Req, 20, 40)
        PostCommand(Req1, false)
        Thread.sleep(50)
        Req1 = Arrays.copyOfRange(Req, 40, 46)
        PostCommand(Req1, false)
        Thread.sleep(50)
    }
    fun ExecuteForceSync(IsFromActivity: Boolean) {
        GetLastHRSync()
        GetLastMainSync()

PostCommand(CommandInterpreter.requestHRHistory(LastHRSync),
false)
        Wait(1000)

PostCommand(CommandInterpreter.SyncTime(Calendar.getInstance()),
false)
        Wait(1000)
        PostCommand(CommandInterpreter.GetMainInfoRequest(),
false)
        Wait(1000)

PostCommand(CommandInterpreter.requestSleepHistory(LastSleepSync
), false)
        Wait(1000)
        if (IsAlarmingTriggered && !IsFromActivity)
AlarmTriggerDecider(0)
    }
    fun ChangeStatus(Text: String) {
        var Text = Text
        val IF = Intent(StatusAction)
        if (HRAnalyzer.isShadowAnalyzerRunning)
            Text += "\nData analyzer is running..."
        IF.putExtra("Data", Text)
        LastStatus = Text
        sendBroadcast(IF)
    }

    fun run() {
        if (Thread.currentThread().name !== "Syncer") return

HRAnalyzer.analyzeShadowMainData(this.Database!!.writableDatabas
e) // TODO test
        while (UartService.instance == null) Wait(3000)
        while (IsActive) {
            if (DeviceControllerActivity.StatusCode < 2) {
                when (DeviceControllerActivity.StatusCode) {
                    -2 -> {
                        if(DeviceControllerActivity.IsActive) {
```

```
Utils.RequestEnableBluetooth(DeviceControllerActivity.instance!!
)
                            if
(Utils.BluetoothEngaging(DeviceControllerActivity.instance!!)) {

DeviceControllerActivity.StatusCode = 0

ChangeStatus(getString(R.string.status_engaging))
                            } else{

ChangeStatus(getString(R.string.offline_mode))
                            }
                        }
                    }
                    -1, 0 -> if
(UartService.instance!!.connect(LockedAddress)) {
                            ChangeStatus("Status :
Connecting...")
                            while
(UartService.instance!!.mConnectionState <
UartService.STATE_CONNECTED) {
                                Thread.sleep(2500)
                            }
                            DeviceControllerActivity.StatusCode
= 1
                        }
                    1 -> {

ChangeStatus(getString(R.string.discovering))
                        if
(UartService.instance!!.mConnectionState <
UartService.STATE_DISCOVERED) {
                            Thread.sleep(3000)

UartService.instance!!.retryDiscovering()
                        } else{
                            DeviceControllerActivity.StatusCode
= 3
                        }
                    }
                }
                continue
            }
            if (!IsAlarmWaiting) CheckForAlarms()
            if (!ServiceObject!!.isBinderAlive && !
IsNotifyServiceAlive(this)) {
                TryForceStartListener(this)
            }
            val h = Handler(Looper.getMainLooper())
            h.post { ExecuteForceSync(false) }
            NextSync = Calendar.getInstance()
```

```
            NextSync!!.add(Calendar.MILLISECOND,
MainSyncPeriodSeconds)
            val SDF = SimpleDateFormat("HH:mm")
            ChangeStatus("Status : Connected. Next Sync Will Be
At " + SDF.format(NextSync!!.time))
            if (!
DatabaseController.getDCObject(this).currentDataBase!!.inTransac
tion() && hardTask == null){
                hardTask = AsyncCollapser()
                hardTask!!.execute()
            }
            Wait(MainSyncPeriodSeconds.toLong())
        }
    }

    inner class LocalBinder : Binder() {
        internal val service: Algorithm
            get() = this@Algorithm
    }
    override fun onBind(intent: Intent): IBinder? {
        Init()
        return mBinder
    }
    override fun onUnbind(intent: Intent): Boolean {
        return super.onUnbind(intent)
    }

    companion object {
        var NextSync: Calendar? = null
        var IsRealTimeSynced = false
        lateinit var SelfPointer: Algorithm
        var BatteryHolder : Int = -1
        var LastHearthRateIncomed : Int = -1
        var LastStepsIncomed : Int = -1
        var LastCcalsIncomed : Int = -1
        var LastStatus       : String = ""

        var MainSyncPeriodSeconds = 310000 // 5`10`` in millis
        val StatusAction = "STATUS_CHANGED"
        private val HRSyncPeriodSeconds: Long = 3600
        private val SleepSencPeriodSeconds: Long = 88000
        private val OptimizationDays: Long = 605000000// +- 7
days in millis TODO some scripts for collapsing data?

        private var MinHR: Int = 0
        private var AvgHR: Int = 0
        private var MaxHR: Int = 0

        var LockedAddress: String? = null

        var ApproachingAlarm: AlarmProvider? = null
        var IsAlarmWaiting = false
```

```
        var IsAlarmingTriggered = false
        var IsAlarmKilled = false

        var IsActive = true
        private var IsInit = false

        private var PhoneListener: PhoneStateListenerBroadcast?
= null
        private var SBIReceiver:
UartServiceBroadcastInterpreter? = null
        fun IsNotifyServiceAlive(context: Context): Boolean {
            val Names =
NotificationManagerCompat.getEnabledListenerPackages(context)
            return Names.contains(context.packageName)
        }
        fun TryForceStartListener(context: Context) {
            val pm = context.packageManager
            pm.setComponentEnabledSetting(ComponentName(context,
NotificationService::class.java),

PackageManager.COMPONENT_ENABLED_STATE_DISABLED,
PackageManager.DONT_KILL_APP)
            pm.setComponentEnabledSetting(ComponentName(context,
NotificationService::class.java),

PackageManager.COMPONENT_ENABLED_STATE_ENABLED,
PackageManager.DONT_KILL_APP)
        }
        private var AlarmFiredIterator = 0
        fun PostCommand(Request: ByteArray, IsForAlarm :
Boolean) {
                if ((IsForAlarm && AlarmFiredIterator%50 == 0)
|| !IsForAlarm) {
                    val RThread =
{ UartService.instance!!.writeRXCharacteristic(Request) }
                    RThread.run
{ UartService.instance!!.writeRXCharacteristic(Request) }
                }
        }
        private fun Wait(MiliSecs: Long) {
            try {
                Thread.sleep(MiliSecs)
            } catch (e: InterruptedException) {
                e.printStackTrace()
            }

        }
    }


    inner class AsyncCollapser: AsyncTask<Void, Void, Void>(){
```

```
        override fun doInBackground(vararg params: Void?): Void?
{

MainRecordsTable.executeDataCollapse(Prefs!!.getLong(MainRecords
Table.SharedPrefsMainCollapsedConst, 0), Prefs!!,
Database!!.currentDataBase!!)
            return null
        }
        override fun onCancelled(result: Void?) {
            hardTask = null
            super.onCancelled(result)
        }
        override fun onCancelled() {
            hardTask = null
            super.onCancelled()
        }
        override fun onPostExecute(result: Void?) {
            hardTask = null
            super.onPostExecute(result)
        }
    }
}
```